AFRL-AFOSR-UK-TR-2014-0020

State-bound estimation for nonlinear systems using randomized mu-analysis

Jongrae Kim

THE UNIVERSITY OF GLASGOW
DIVISION OF BIOMEDICAL ENGINEERING
UNIVERSITY AVENUE
GLASGOW G12 8QQ UNITED KINGDOM

EOARD Grant 13-3029

Report Date: April 2014

Final Report from 28 February 2013 to 27 February 2014

Air Force Research Laboratory
Air Force Office of Scientific Research
European Office of Aerospace Research and Development
Unit 4515, APO AE 09421-4515

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
30 April 2014	Final Report	28 February 2013 – 27 February 2014

4. TITLE AND SUBTITLE

State-bound estimation for nonlinear systems using randomized mu-analysis

5a. CONTRACT NUMBER

FA8655-13-1-3029

5b. GRANT NUMBER

Grant 13-3029

5c. PROGRAM ELEMENT NUMBER

61102F

6. AUTHOR(S)

Jongrae Kim

5d. PROJECT NUMBER

5d. TASK NUMBER

5e. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

THE UNIVERSITY OF GLASGOW
DIVISION OF BIOMEDICAL ENGINEERING
UNIVERSITY AVENUE
GLASGOW G12 8QQ UNITED KINGDOM

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
Unit 4515
APO AE 09421-4515

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/AFOSR/IOE (EOARD)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

AFRL-AFOSR-UK-TR-2014-0020

12. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution A: Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Developing state bound estimation algorithms for nonlinear systems has been of high importance in robustness analysis of dynamic systems. For many cases, Monte-Carlo simulation might be the only tool to estimate these bounds for a general type of nonlinear systems. The required number of simulations for a tight bound, however, would be very large and it might be impossible to complete within a given computation time. mu-formulation for state bounds transforms the bound estimation problem to a singularity problem and the singular problem is solved using a randomized optimization approach. The performance of the algorithms is demonstrated by multi-dimensional Rosenbrock function; simple discrete system; large-scale biological system; hybrid system; and navigation error propagation for underwater vehicle. For a given error tolerance of the bounds, a formula to calculate the required number of sampling in the algorithms is provided. Because of the inherent complexity of general nonlinear optimization problems, the required sampling number increases very fast as the problem dimension increases. The suggested algorithms would produce, however, tighter estimation faster than random blind search. In addition, for exploiting parallel computation architecture, the suggested algorithms could be the solution for real-time robustness analysis in the future.

15. SUBJECT TERMS

EOARD, robust control, mu-analysis, state bound estimation, nonlinear robustness analysis

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18, NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Kevin Bollino
UNCLAS	UNCLAS	UNCLAS	SAR	76	19b. TELEPHONE NUMBER *(Include area code)* +44 (0)1895 616163

State-bound estimation for nonlinear systems using randomised μ-analysis

Grant Number: FA8655-13-1-3029

Grant Period: 1st May 2013 - 30H April 2014

Jongrae Kim

Primary: Division of Biomedical Engineering
Secondary: Aerospace Sciences
University of Glasgow, Glasgow G12 8QT
<Jongrae.Kim@glasgow.ac.uk>

30th April 2014

Summary

Developing state bound estimation algorithms for nonlinear systems has been of high importance in robustness analysis of dynamic systems. For many cases, Monte-Carlo simulation might be the only tool to estimate these bounds for a general type of nonlinear systems. The required number of simulations for a tight bound, however, would be very large and it might be impossible to complete within a given computation time. μ-formulation for state bounds transforms the bound estimation problem to a singularity problem and the singular problem is solved using a randomised optimisation approach. The performance of the algorithms are demonstrated by multi-dimensional Rosenbrock function; simple discrete system; large-scale biological system; hybrid system; and navigation error propagation for underwater vehicle. For a given error tolerance of the bounds, a formula to calculate the required number of sampling in the algorithms is provided. Because of the inherent complexity of general nonlinear optimisation problems, the required sampling number increases very fast as the problem dimension increases. The suggested algorithms would produce, however, tighter estimation faster than random blind search. In addition, exploiting parallel computation architecture the suggested algorithms could be the solution of real-time robustness analysis in future.

The following two conference papers are accepted to 19th IFAC World Congress 2014, 24th- 29th August 2014, Cape Town, South Africa:

1. Kim, J., Kishida, M., and Bates, D.G., "State bounds estimation for nonlinear systems using mu-analysis"

2. Kim, J., Park, Y., Lee, S., and Lee, Y.K., "Underwater glider navigation error compensation using sea current data"

Contents

Chapter 1

Introduction

Robustness is the capability of systems maintaining their functions while internal and external disturbances produce adverse effects on the performance. Robustness analysis is an indispensable step in designing engineering systems [1, 2, 3]. It is also considered as one of the most important aspects in analysing biological systems [4, 5, 6, 7, 8, 9, 10]. Systematic approach to model many interactions and analyse noisy measurement data is one of the highly preferable ways in improving understanding of complex systems [11, 12].

Recently, some of biological system models are described with hundreds of states and parameters [13, 14]. And, such large-scale systems provide information that was not available with small scale models. Similarly, engineering systems have been evolving to be more and more complex in order to accomplish improved performance while attaining stronger robustness towards various uncertain environment. This tendency will continue in future for satisfying much demanding design specifications and it is important, hence, to have efficient numerical methods to analyse large-scale systems.

Structured singular value or μ-analysis has been one of the most successful tools for robustness evaluation [15, 16, 17, 18]. Even though the computational complexity of μ-analysis was acknowledged at the beginning state of the concept introduced, it has been successfully used for many practical systems design and

analysis.

The computational complexity of μ-analysis is proved in [19] by transforming μ problem to a corresponding optimisation problem that is known to be Non-deterministic Polynomial-time hard (NP-hard). As far as NP$\neq$P, the computational complexity is the fundamental obstacle that cannot be overcome by any algorithms without introducing conservatism in design and analysis. Practically, however, μ-analysis algorithm produced many useful results. In addition, recently, it is further extended to solve some class of optimisation problems [20]. This is an inverse interpretation of the formulation in [19].

The application in [20] is calculating state bounds for polynomial nonlinear discrete systems, where initial states and parameters in the systems are described by uncertain bounds. Then, the state maximum and minimum bounds are calculated using μ upper and lower bounds algorithms. As long as the nonlinearity appears in polynomial, the uncertainties can be decoupled from the known parts and the system can be described in Linear Fractional Transformation (LFT) [19, 21]. Once LFT form is obtained, some powerful numerical tools that provide the upper and the lower bounds of μ can be used, for example, μ toolbox or Robust Control toolbox in MATLAB [1].

Conservatism of the calculated bounds and the requirement for the uncertainty structures for enabling LFT format are two main obstacles for the state bounds algorithm to be further extended its applications. μ-analysis problem is interpreted in geometrical point of view in [22] and it enables one to use random sampling approach to obtain the bounds. Later, it is further lifted the requirement of LFT-transformation, i.e. LFT-free μ-analysis [23]. It is, hence, a natural fusion that combining state bounds estimation and LFT-free μ-analysis.

In this research calculating the bounds of a polynomial function using the skewed μ-analysis framework presented in [20] is extended to a general type of nonlinear systems including discontinuous and non-smooth nonlinear functions using a random sampling method [23].

This report is organised as follows: Firstly, state bounds estimation prob-

lem is formulated as LFT-free μ-analysis and state worst-bounds algorithms are presented; Secondly, the performance of the algorithms is demonstrated using various examples: *hello world* example, oscillatory discrete system, large-scale biological system, hybrid system, and underwater glider navigation error propagation, where some of the examples are used the algorithm parallelised to run on Graphical Processing Unit (GPU); Finally, the conclusions are presented.

Chapter 2

Methods, Assumptions, and Procedures

2.1 Bounds estimation in LFT-formulation

A nonlinear dynamical system is given by

$$\dot{x} = f(x, p), \tag{2.1}$$

where $x_0 = x(0)$ is the initial condition at time $t = 0$, $\dot{x}$ is the derivative of x by time, t , x is the state vector, which is an element of the n_x-dimensional real number set, $\mathbb{R}^{n_x}$, n_x is a positive integer, p is the uncertain parameters, which is an element of n_p-dimensional real number set, $\mathbb{R}^{n_p}$, n_p is a positive integer, and $f(x, p)$ is a nonlinear function in x and p, which might have discontinuous and/or non-smooth parts. Many dynamic systems will fall into this category, for example, sun tracking controller for UKube-1 using magnetic torquer [24], spacecraft attitude control using thruster [25], auto-tuning control mechanism for industrial applications [26].

For the well-posedness of the problem, $f(x, p)$ is assumed to have a unique solution for the nonlinear differential equation. For a chosen positive real num-

ber, Δt, a transition function $\psi(,)$ is defined to satisfy the following:

$$x_{k+1} = \psi(x_k, p) \tag{2.2}$$

where $x_k = x(k\Delta t)$ for $k = 0, 1, 2, \ldots$. The transition function can be obtained by a standard numerical integrator such as Runge-Kutta method.

The main problem is finding the worst-bounds for the maximum and the minimum of a function, $\phi(x_k)$, for the given bounds for x_0 and p as follows:

Problem 2.1.1 *(Bounds estimation) For a real-valued scalar function, $\phi(x_k, p)$, obtain $\underline{\phi}_{\min}$, $\bar{\phi}_{\min}$, $\underline{\phi}_{\max}$ and $\bar{\phi}_{\max}$ in the following inequalities:*

$$\underline{\phi}_{\min} \leq \underline{\phi} \leq \bar{\phi}_{\min}, \tag{2.3a}$$

$$\underline{\phi}_{\max} \leq \bar{\phi} \leq \bar{\phi}_{\max}, \tag{2.3b}$$

where $\underline{\phi}$ and $\bar{\phi}$ are the unknown true bounds satisfying $\underline{\phi} \leq \phi(x_k, p) \leq \bar{\phi}$ for all $\underline{x}_0 \leq x_0 \leq \bar{x}_0$, and $\underline{p} \leq p \leq \bar{p}$. That is, $\bar{\phi}$ is the minimum upper bound and $\underline{\phi}$ is the maximum lower bound.

Remark 2.1.2 *The real-valued scalar function, $\phi(x_k, p)$, could be the position bounds of a group of debris in space, the miss distance bounds of missile, or the navigation error bounds of mobile robot.*

In the above, all the upper and the lower bounds for $\phi(x_k, p)$, x_0 and p are assumed to be finite. Systems having a finite escape time to either positive or negative infinity are excluded in this study

Let $\phi(x_k, p) = x_k$ in Problem 2.1.1. Then, the problem becomes *state bound estimation problem* for the give uncertain ranges of the initial condition and the parameters in the system. For the case that $\phi(x_k, p)$ is a polynomial function in x_k and p, it can be transformed into LFT form and existing μ bounds algorithms can be directly used to obtain the bounds. See the below example.

Example 2.1.3 *(Special Case) Find the polynomial expression for the state bounds at $t = 2$, i.e. $x(2)$, of the following dynamic system:*

$$\dot{x} = -(1+p)x \tag{2.4}$$

where $0 \leq x(0) \leq 1$ *and* $-\frac{1}{2} \leq p \leq \frac{1}{2}$.

sol) The transition function is given by

$$\phi[x(t), p] = x(t) = \psi[x(t), p] = e^{-(1+p)t}x(0)$$

At $t = 2$,

$$\phi[x(2), p] = e^{-(1+p)2}x(0) = e^{-2}e^{-2p}x(0)$$

Using the Taylor series expansion, the uncertain exponential function is given by

$$\phi[x(2), p] = e^{-2}\left(1 - 2p + \frac{4p^2}{2!} - \frac{8p^3}{3!} + \dots\right)x(0)$$

As $|p| \leq 0.5$, ignore the higher order terms in p as follows:

$$\phi[x(2), p] \approx e^{-2}\left(1 - 2p + p^2\right)x(0) \tag{2.5}$$

Let the nominal values for $x(0)$ and p be $1/2$ and 0, which are the midpoints of the given uncertain boundaries, respectively. Using the midpoints, each can be written as follows:

$$x(0) = \frac{1}{2} + \frac{1}{2}\delta_x \tag{2.6a}$$

$$p = 0 + \frac{1}{2}\delta_p \tag{2.6b}$$

where $-1 \leq \delta_x \leq 1$ and $-1 \leq \delta_p \leq 1$.

Substituting (2.6) into (2.5)

$$\phi[x(2), p] \approx e^{-2}\left(1 - \delta_p + \frac{\delta_p^2}{4}\right)\left(\frac{1}{2} + \frac{1}{2}\delta_x\right) \tag{2.7}$$

Hence, this is now in a polynomial function and it can be transformed into LFT-form. Details about the transformation of the special case can be found in [20].

□

On the other hand, the Problem 2.1.1 is not possible to transform into a polynomial expression, in general, without introducing some approximation error, which might cause significant conservatism in the estimated bounds. In the following, the same idea obtaining the state bounds using μ-formulation in [20] is extended without introducing any polynomial approximation.

Let x_0 and p in Problem 2.1.1 be the following form:

$$x_0 = x_c + W_x\delta_x, \tag{2.8a}$$

$$p = p_c + W_p\delta_p, \tag{2.8b}$$

where

$$W_x = \text{diag}\begin{bmatrix} w_{x_1} & w_{x_2} & \dots & w_{x_{n_x}} \end{bmatrix}, \tag{2.9a}$$

$$W_p = \text{diag}\begin{bmatrix} w_{p_1} & w_{p_2} & \dots & w_{p_{n_p}} \end{bmatrix}, \tag{2.9b}$$

diag[...] is the diagonal matrix whose diagonal terms are given in the bracket, $w_{x_i} = (\bar{x}_0 - \underline{x}_0)/2$, $w_{p_j} = (\bar{p} - \underline{p})/2$, x_c and p_c are defined such that

$$-1 \leq \delta_{x_i} \leq 1 \tag{2.10a}$$

$$-1 \leq \delta_{p_j} \leq 1 \tag{2.10b}$$

for $i = 1, 2, \dots, n_x$ and $j = 1, 2, \dots, n_p$.

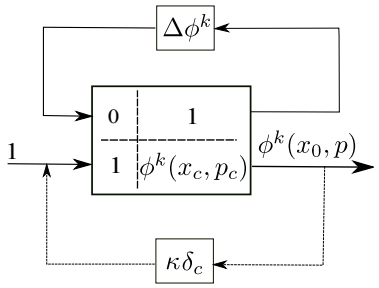

Figure 2.1: Pseudo-LFT form

Example 2.1.4 *In the system shown in Example 2.1.3, the original function without the approximation is*

$$\phi[x(2), p] = e^{-2}e^{-2p}x_0$$

where $p = p_c + w_p\delta_p$, $x_0 = x_c + w_x\delta_x$, $p_c = 0$, $w_p = 1/2$, $x_c = 1/2$, $w_x = 1/2$, *and* $-1 \leq \delta_p \leq 1$, $-1 \leq \delta_x \leq 1$.

□

Define

$$\Delta\phi^k(\delta_x, \delta_p) := \phi^k(x_0, p) - \phi^k(x_c, p_c) \tag{2.11}$$

where

$$\phi^k(\cdot, p) := \phi[\underbrace{\psi \circ \psi \circ \psi \circ \ldots \circ \psi}_{k\text{-times}}(\cdot, p), p] \tag{2.12}$$

and

$$\phi^2(\cdot, p) = \phi[\psi \circ \psi(\cdot, p), p] = \phi\left\{\psi[\psi(\cdot, p), p], p\right\} \tag{2.13}$$

Pseudo-LFT form as shown in Figure 2.1 is to be constructed. It is called pseudo-LFT as it is in the LFT format but it can be only evaluated for a fixed δ_x and δ_p. In the standard LFT-formulation, $\Delta\phi^k$ is a constant matrix with a structure. On the other hand, in the pseudo-LFT formulation, it is a varying

vector evaluated on a specific combination of δ_x and δ_p.

From the pseudo-LFT shown in Figure 2.1 and the equivalency between μ-bounds and the optimisation problem shown in [19], the maximum of $|\phi^k(x_0, p)|$ is bounded above as

$$\max |\phi^k(x_0, p)| \leq \frac{1}{\kappa^*}, \tag{2.14}$$

where κ^* is the minimum κ satisfying the following singular condition:

$$|I_2 - N\Delta| = 0, \tag{2.15}$$

$|\cdot|$ is the determinant of matrix, I_2 is the 2×2 identify matrix,

$$N = \begin{bmatrix} 0 & 1 \\ 1 & \phi^k(x_c, p_c) \end{bmatrix}, \tag{2.16a}$$

$$\Delta = \begin{bmatrix} \Delta\phi^k & 0 \\ 0 & \kappa\delta_c \end{bmatrix}, \tag{2.16b}$$

$|\delta_c| = |\delta_R + \delta_I j| \leq 1$, δ_R and δ_I are the real numbers whose magnitude is less than or equal to 1, and $j = \sqrt{-1}$. Note that κ is a positive number satisfying the singular condition.

The singularity condition is expanded

$$\begin{aligned} |I_2 - N\Delta| &= \left| I_2 - \begin{bmatrix} 0 & \kappa\delta_c \\ \Delta\phi^k & \kappa\phi^k(x_c, p_c)\delta_c \end{bmatrix} \right| = \begin{vmatrix} 1 & -\kappa\delta_c \\ -\Delta\phi^k & 1 - \kappa\phi^k(x_c, p_c)\delta_c \end{vmatrix} \\ &= 1 - \kappa\phi^k(x_c, p_c)\delta_c - \kappa\Delta\phi^k\delta_c \\ &= \left\{1 - \kappa\left[\phi^k(x_c, p_c) + \Delta\phi^k\right]\delta_R\right\} - \kappa\left[\phi^k(x_c, p_c) + \Delta\phi^k\right]\delta_I j = 0, \end{aligned} \tag{2.17}$$

where the real part and the imaginary part must be equal to zero at the same

time for the matrix to be singular. The real and the imaginary parts are equal to zero as follows:

$$\Re\left(|I_2 - N\Delta|\right) = 1 - \kappa\left[\phi^k(x_c, p_c) + \Delta\phi^k\right]\delta_R = 0, \tag{2.18a}$$

$$\Im\left(|I_2 - N\Delta|\right) = -\kappa\left[\phi^k(x_c, p_c) + \Delta\phi^k\right]\delta_I = 0, \tag{2.18b}$$

where $\Re(\cdot)$ and $\Im(\cdot)$ are the real part and the imaginary part of the argument, respectively. Notice that $\phi^k(x_c, p_c) + \Delta\phi^k$ is equal to $\phi^k(x_0, p)$ by the definition and it is not equal to zero, in general.

Hence, δ_I must be equal to zero for the singularity condition. Therefore, the following is the only case that both the real and the imaginary parts are equal to zero: $1 - \kappa\left[\phi^k(x_c, p_c) + \Delta\phi^k\right]\delta_R = 0$ and $\delta_I = 0$, i.e., the imaginary value of δ_c is always equal to zero.

Take the absolute value of (2.18a) after moving the second term into the right hand side of the equation as follows: $1 = \left|\kappa\ \phi^k(x_0, p)\delta_R\ \right|$, and it becomes

$$\kappa = \frac{1}{|\phi^k(x_0, p)\delta_R\ |}. \tag{2.19}$$

Finally, the singular problem for the bound estimation is formulated:

Problem 2.1.5 *(Singular problem) Find the minimum κ, i.e. κ^*, satisfying the following:*

$$\kappa^* = \frac{1}{\max|\phi^k(x_0, p)\delta_R|} = \frac{1}{\max|\phi^k(x_0, p)|} \tag{2.20}$$

where $|\delta_R| \leq 1$.

This looks trivial and it does not seem to help to find the bounds for $\phi^k(x_0, p)$. In the next section, bounds algorithms using a random sampling approach will be presented.

Example 2.1.6 *For the system shown in Example 2.1.4, the singular problem is given by $\kappa^* = 1/\max|e^{-2}e^{-2p}x_0|$.* □

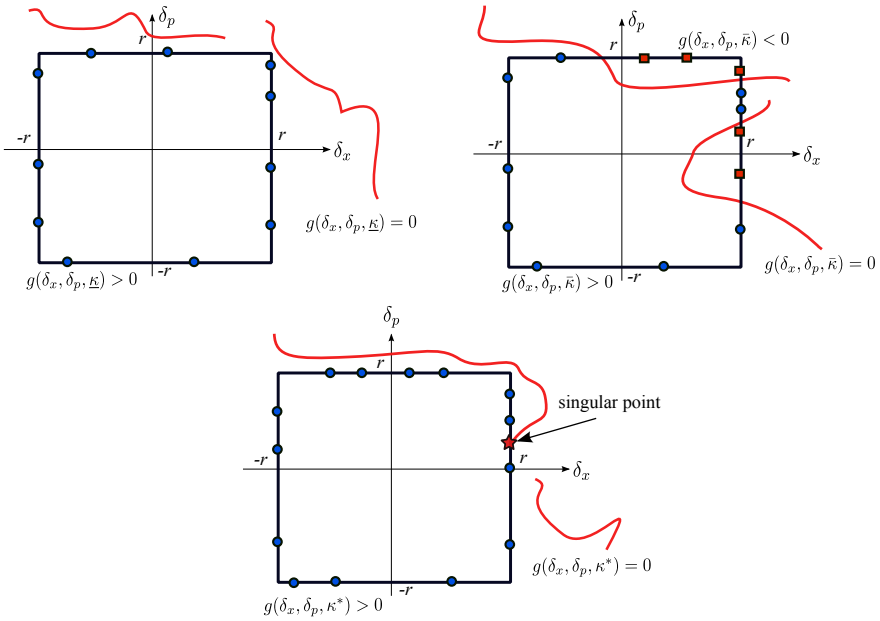

Figure 2.2: Sign changes along the uncertain box boundary: as $\underline{\kappa}$ is smaller than κ^*, all signs of $g(\delta_x, \delta_p, \underline{\kappa})$ for the samples at the boundary are positive; on the other hand, when $\bar{\kappa}$ is greater than κ^*, there are red box samples and blue circle samples whose sign of $g(\delta_x, \delta_p, \bar{\kappa})$ is positive and negative, respectively.

2.2 Worst Bounds Estimation Algorithms

Define

$$g(\delta_x, \delta_p, \kappa) := 1 - \kappa |\phi^k(x_0, p)| = 1 - \kappa \left|\phi^k(x_c + W_x \delta_x, p_c + W_p \delta_p)\right| \qquad (2.21)$$

where $-r \leq \delta_x \leq r$, $-r \leq \delta_p \leq r$, $\kappa > 0$, and $r \in (0, 1]$. Note that $g(\cdot, \cdot, \cdot)$ could be discontinuous as $\phi^k(\cdot, \cdot)$ could be discontinuous. As shown in Figure 2.2, there are three cases:

- for $\kappa < \kappa^*$, having one type of samples, whose signs are all positive,
- for $\kappa > \kappa^*$ having two types of samples, whose signs are positive or negative,
- for $\kappa = \kappa^*$ having at least one singular point, where $g(\delta_x, \delta_p, \kappa)$ is equal to zero.

Hence, whenever for a fixed κ, if two sign combinations are found, it becomes the upper bound for κ^*, i.e., $\bar{\kappa}$. Similarly, if only positive signs are found, it becomes the lower bound, $\underline{\kappa}$.

Example 2.2.1 *For the system shown in Example 2.1.6, the function, $g(\delta_x, \delta_p, \kappa)$, is given by*

$$g(\delta_x, \delta_p, \kappa) := 1 - \kappa \left| e^{-2} e^{-\delta_p} \left(\frac{1}{2} + \frac{1}{2}\delta_x \right) \right|$$

□

The bound estimation problem is given by

Problem 2.2.2 *(κ^*-bounds estimation) For the following real-valued scalar function:*

$$g(\delta_x, \delta_p, \kappa) = 1 - \kappa \left| \phi^k(x_c + W_x\delta_x, p_c + W_p\delta_p) \right|, \tag{2.22}$$

obtain the upper ($\bar{\kappa}$) and lower ($\underline{\kappa}$) bounds as follows:

$$\underline{\kappa} \leq \kappa^* \leq \bar{\kappa} \tag{2.23}$$

Remark 2.2.3 *The upper bound, $\bar{\kappa}$, is a deterministic bound as we found the negative sign but $\underline{\kappa}$ is probabilistic as it has always some danger to be failed depending on the number of samples checked on the boundary. A safety factor could be introduced in order to reduce the probability for the upper bound to be failed.*

Remark 2.2.4 *Once the bounds for κ^* are obtained, the bounds for $|\phi^k(\cdot,\cdot)|$, is obtained such that $\underline{\phi}_{\min}$ (or $\underline{\phi}_{\max}$) is equal to $1/\bar{\kappa}$ and $\bar{\phi}_{\min}$ (or $\bar{\phi}_{\max}$) is equal to $1/\underline{\kappa}$.*

How to resolve whether they are the bounds for the minimum or the maximum will be presented later. In the following, bounds algorithms for a fixed

radius uncertain space, r, which is greater than 0 and less than or equal to 1, are presented and an algorithm for sweeping r in $(0, 1]$ will be presented.

2.2.1 Fixed uncertain space radius: $r \in (0, 1]$

First part of the algorithm is for calculating the pre-κ bound.

Algorithm 2.2.5 *Pre-$\underline{\kappa}$ estimation*

1. *Set N, the number of samples along the face of the uncertain box shown in Figure 2.2, where the size of the box is r measured in terms of ∞-norm.*
2. *Set the initial boundary for κ such that $\epsilon \leq \kappa \leq E$, where ϵ could be the smallest positive number and E could be the largest positive number that can be expressed in the computer.*
3. *Set the tolerance, ε, for the magnitude of the interval, $[\epsilon, E]$, i.e. $E - \epsilon$*
4. *Set initial guess of κ, equal to $(\epsilon + E)/2$*
5. *For $i = 1$ to N*
   - *Evaluate $g(\delta_x, \delta_p, \kappa)$ for the given i-th sample of δ_k and δ_p*
   - *if $g(\delta_x, \delta_p, \kappa) < 0$, then replace E by κ and break the for-loop, else continue the for-loop*
6. *If $i = N$ and $g(\delta_k, \delta_p, \kappa)$ for all samples are positive, then replace ϵ by κ.*
7. *If $E - \epsilon$ is smaller than ε, then declare $\underline{\kappa}_p = E$ and stop. Otherwise, go to step #4*

Note that $\underline{\kappa}_p$ from the pre-$\underline{\kappa}$ estimation algorithm does not need to be tight as long as it is smaller than κ^*. The reason to calculate $\underline{\kappa}_p$ using the above algorithm before calculating bounds for $\phi^k(x_0, p)$ is that the value of $\phi^k(x_0, p)$ can be positive and negative and the definition of κ^* in (2.20) is given in terms of the absolute value of $\phi^k(x_0, p)$. On the other hand, if the sign of all possible values of ϕ^k is either only positive or negative, this step can be skipped.

In order to estimate the bounds for the maximum $\phi^k(x_0, p)$, the following is defined:

$$\bar{g}(\delta_x, \delta_p, \kappa) := 1 - \kappa|\phi^k(x_0, p) + s/\underline{\kappa}_p| \tag{2.24}$$

where s is the shifting factor strictly greater than 1. As $1/\underline{\kappa}_p > |\phi^k(x_0, p)|$ can be guaranteed only in a probabilistic sense, the safety factor will make sure the terms inside the absolute sign be positive. Then, the maximum of $|\phi^k(x_0, p) + s/\underline{\kappa}_p|$ occurs at $\bar{\phi}^k(x_0, p) + s/\underline{\kappa}_p$.

Algorithm 2.2.6 *$\bar{\phi}$-bounds estimation Algorithm*

1. *Run the pre-$\underline{\kappa}$ estimation algorithm after replacing $g(\delta_x, \delta_p, \kappa)$ by $\bar{g}(\delta_x, \delta_p, \kappa)$*
2. *Set $\underline{\kappa} = \epsilon$ and $\bar{\kappa} = E$*
3. *Declare $\underline{\phi}_{\max} = 1/\bar{\kappa} - s/\underline{\kappa}_p$ and $\bar{\phi}_{\max} = s_f/\underline{\kappa} - s/\underline{\kappa}_p$, where s_f is the safety factor greater than 1. The smaller s_f returns tighter bounds with greater danger to be failed. The larger s_f returns conservative bounds with less probability to be failed.*

Similarly, the bounds for the minimum is obtained by defining

$$\underline{g}(\delta_x, \delta_p, \kappa) := 1 - \kappa|\phi^k(x_0, p) - s/\underline{\kappa}_p|, \tag{2.25}$$

which make sure the maximum occurs at the minimum of ϕ^k, and executing the following algorithm:

Algorithm 2.2.7 *$\underline{\phi}$-bounds estimation Algorithm*

1. *Run the pre-$\underline{\kappa}$ estimation algorithm after replacing $g(\delta_x, \delta_p, \kappa)$ by $\underline{g}(\delta_x, \delta_p, \kappa)$*
2. *Set $\underline{\kappa} = \epsilon$ and $\bar{\kappa} = E$*
3. *Declare $\underline{\phi}_{\min} = 1/\bar{\kappa} + s/\underline{\kappa}_p$ and $\bar{\phi}_{\min} = s_f/\underline{\kappa} + s/\underline{\kappa}_p$*

Remark 2.2.8 *The bounds calculated by the above algorithms are for a fixed r. Hence, the global bounds must be obtained by finding the maximum bounds among the ones for different r. However, the lower bound, the worst scenario found so far, is always valid and the issue is how close it will be to the true bound.*

2.2.2 Sweeping uncertain space: $r \in (0, 1]$

The bounds calculated in the previous section is for a fixed r in $(0, 1]$, i.e., $\underline{\kappa}(r)$ and $\bar{\kappa}(r)$, where the argument indicates that the bounds are a function of r. To find the global bounds, the algorithms in the previous must be performed in r-space.

Algorithm 2.2.9 *Global bounds*

1. *Set N_r, the number of samples in $(0, 1)$.*
2. *Generate N_r uniformly distributed random numbers in $(0, 1)$.*
3. *Initialise the set, $R = \{r_0, r_1, r_2, \ldots, r_{N_r}, r_{N_r+1}\}$, where $r_0 = 0$, r_i for $i = 1, 2, \ldots, N_r$, are the random numbers generated in #2, and $r_{N_r+1} = 1$.*
4. *For all $r_s \in R$ perform the following:*
   - *Run Algorithms 2.2.5, and 2.2.6 for $\bar{\phi}$ or 2.2.7 for $\underline{\phi}$.*
   - *Using $\underline{\kappa}(r_s)$ and $\bar{\kappa}(r_s)$, calculate the bounds for ϕ^k, i.e., $\bar{\phi}_{\max}$, $\underline{\phi}_{\max}$ or $\bar{\phi}_{\min}$, $\underline{\phi}_{\min}$.*
5. *For r_i in R, compare the bounds difference with either r_{i-1} or r_{i+1} in R, generate new r_i towards the direction where the bounds improve.*
6. *If $|r_{N_r+1} - r_0|$ is less than a tolerance, then stop. Otherwise, go to #4 with the updated R.*

It is worth to note that the suggested algorithm is different from the blind Monte-Carlo random search. The search performs through random samples but

each iteration it converges towards a solution. As a result, the search space reduces every iteration.

Remark 2.2.10 *The total computational cost depends on how many times the algorithms are executed for different r in $(0, 1]$. If the upper bounds could be provided with less computation, the proposed algorithm could be used only for finding lower bounds.*

The proposed four algorithms in the above are *embarrassingly parallel*, i.e. all sampling evaluations are independent from each other and it can be easily implemented on parallel computer architectures. Therefore, it is a perfect problem to be solved on multi-core, distributed parallel computer nodes and GPU. For some examples presented in the next chapter, the sampling evaluation part of the algorithms is running on NVIDIA Tesla C2050 GPU, which has 449 cores and the maximum number of threads per block is 1024. The algorithms on the GPU is implemented using CUDA-GPU [27].

2.3 Probabilistic Properties of the Algorithms

A probabilistic guarantee of the performance for the algorithm finding $\bar{\phi}$ is to be derived using the Chernoff bound [28]. Similar derivation can be obtained for the other parts of the algorithms. If the number of samples, N, satisfies the following inequality:

$$N \geq \frac{1}{2\epsilon_c^2} \log \frac{2}{\delta_c} \tag{2.26}$$

then the following is true:

$$\text{Probability}\left\{|p(r) - \hat{p}(r)| \leq \epsilon_c\right\} \geq (1 - \delta_c) \tag{2.27}$$

for $\epsilon_c \in (0,\, 1)$ and $\delta_c \in (0,\, 1)$, where

$$p(r) = \frac{\sum \mathcal{V}_i}{\mathcal{V}_T}, \tag{2.28}$$

$\mathcal{V}_i$ is the connected volume, of which volume size is assumed to be strictly greater than zero, where all values of ϕ^k in the volume are greater than the maximum lower bound found by the algorithm, $\mathcal{V}_T$ is the total volume of the search space,

$$\hat{p}(r) = \frac{n_l}{N}, \tag{2.29}$$

and n_l is the number of samples, which belongs to $\mathcal{V}_i$. Based on these, we can prove the following theorem:

Theorem 2.3.1 *If the number of samples is equal to the Chernoff bound,* (2.26) *and*

$$p(r) \geq \frac{1}{N} + \sqrt{\frac{1}{2N} \log \frac{2}{\delta_c}} \ , \tag{2.30}$$

then the probability of finding the larger lower bound than the current one using the algorithm is greater than $(1 - \delta_c)$.

Proof. Let N equal to the Chernoff bound in (2.26),

$$\epsilon_c = \sqrt{\frac{1}{2N} \log \frac{2}{\delta_c}}. \tag{2.31}$$

From (2.27),

$$\text{Probability}\left\{\max\left[0,\, p(r) - \epsilon_c\right] \leq \hat{p}(r) \leq \min\left[1,\, p(r) + \epsilon_c\right]\right\} \geq 1 - \delta_c \tag{2.32}$$

Substituting (2.29) and (2.31) into (2.32), then the probability of the following inequality hold is greater than $1 - \delta_c$:

$$\max\left[0, p(r) - \sqrt{\frac{1}{2N} \log \frac{2}{\delta_c}}\right] \leq \frac{n_l}{N} \leq \min\left[1, p(r) + \epsilon_c\right] \tag{2.33}$$

By substituting the lower bound for $p(r)$, (2.30), into (2.33),

$$\text{Probability}\left\{N \max\left(0, \frac{1}{N}\right) \leq n_l\right\} = \text{Probability}\left\{1 \leq n_l\right\} \geq 1 - \delta_c \tag{2.34}$$

where Probability$\{1 \leq n_l\}$ is the probability of finding at least one sample, whose ϕ^k is greater than the current lower bound. □

Note that if the true probability is in the range:

$$0 < p(r) < \frac{1}{N} + \sqrt{\frac{1}{2N} \log \frac{2}{\delta_c}}, \tag{2.35}$$

then the bound in (2.34) becomes

$$\text{Probability}\,\{0 \leq n_l\} \geq 1 - \delta_c \tag{2.36}$$

i.e. the lower bound is changed from 1 to 0 in the left hand side of bracket. As a result, in this case we cannot say anything about the probability for improving the lower bound.

Chapter 3

Results and Discussion

3.1 "*Hello World*" Example

Consider the following ϕ^k, which is called Rosenbrock function:

$$\phi^k(p) = 100\left(p_2 - p_1^2\right)^2 + (1 - p_1)^2, \tag{3.1}$$

where it is assumed that there is no uncertainty in the initial state, and

$$-\pi \leq p_i \leq \pi\sqrt{2} \tag{3.2}$$

for $i = 1$ and 2. Normalise p_i such that

$$p_i = p_{c_i} + w_{p_i}\delta_{p_i}, \tag{3.3}$$

where

$$w_{p_i} = \frac{\pi\sqrt{2} - \pi}{2}, \tag{3.4a}$$

$$p_{c_i} = -\pi + \frac{\pi\sqrt{2} - \pi}{2}, \tag{3.4b}$$

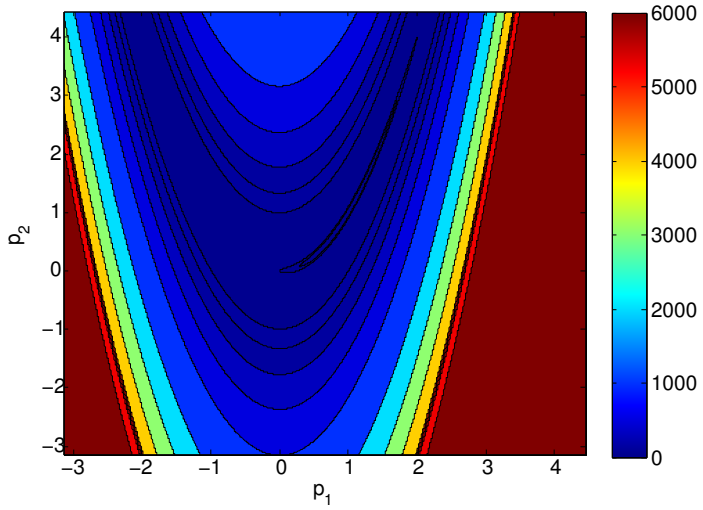

Figure 3.1: Rosenbrock function

and

$$W_p = \begin{bmatrix} w_{p_1} & 0 \\ 0 & w_{p_2} \end{bmatrix}. \tag{3.5}$$

The function, (3.1), has one global minimum at $p_1 = 1$ and $p_2 = 1$, whose function value is zero, and one global maximum at $p_1 = \pi\sqrt{2}$ and $p_2 = -\pi$, whose function value is about 52,365.

For Algorithm 2.2.5, Pre-κ estimation, $N = 1,000$, $\epsilon = 1 \times 10^{-12}$, $E = 1 \times 10^{12}$, $\varepsilon = 0.001$, and the calculated κ_p is equal to 1/52364. For Algorithms 2.2.6 and 2.2.7, $s_f = 1 + 10^{-6}$, $s = 1.2$, ϵ and E are the same as ones in Pre-κ estimation. Finally, these have to be run for different r in Algorithm 2.2.9, where $N_r = 21$ and the tolerance for $|r_{N_r+1} - r_0|$ is 1×10^{-6}. The MATLAB programmes can be found in Appendix A.

From Algorithm 2.2.6, $\underline{\phi}^k_{\max} \approx 52,364.9$ and $\bar{\phi}^k_{\max} \approx 52,365.0$, where the true maximum is about 52,365. On the other hand, from Algorithm 2.2.7, $\underline{\phi}^k_{\min} \approx -0.00001$ and $\bar{\phi}^k_{\min} \approx 0.063$, where the true minimum is equal to 0. Because of the probabilistic nature of the upper bounds, they cannot be trusted with 100% confidence. However, the lower bounds are extremely close to the

Table 3.1: Bounds for the minimum of high-dimensional Rosenbrock function, where the true minimum, $\phi_{\min}$, is equal to 0.

n	2	3	4	5	6
$\bar{\phi}_{\min}$	0.063	0.0906	0.1877	0.2513	-0.0664
$\underline{\phi}_{\min}$	-0.00001	-0.00046	-0.00845	-0.0056	-0.2539

true values in this case.

High-dimensional Rosenbrock function can be defined as

$$\phi^k(p) = \sum_{i=1}^{n-1} 100 \left(p_{i+1} - p_i^2\right)^2 + (1 - p_i)^2 \tag{3.6}$$

where n is the dimension of the uncertain space. The global minimum occurs at all $p_i = 1$ for $i = 1, 2, \ldots, n$ and the minimum is zero. For different n, the bounds for the minimum are shown in Table 3.1. The gap between the upper and the lower bounds becomes larger as the dimension of the uncertain parameter space being larger. It eventually fails at $n = 6$, where the upper bound is smaller than 0. To resolve this, the number of samples, N, is increased from 1000 to 2000. And, the bounds obtained is $\bar{\phi}_{\min} = 0.187$ and $\underline{\phi}_{\min} = -0.0683$.

The relation between the bounds gap and the specified N is not known in general. That is, how N must be chosen in order to make sure that the bounds is correct is unknown. In addition, the required N increases exponentially as the search space dimension increases. For example, from (2.27), N equal to 2000 gives 99.99% confidence in finding $p(r)$ greater than 0.05%. However, $p(r)$ around the global minimum reduces quickly as the search dimension increases. Practically, up to a certain range of n, it can be effectively overcome with increased samples and by parallelising the algorithms over distributed/GPU processing units.

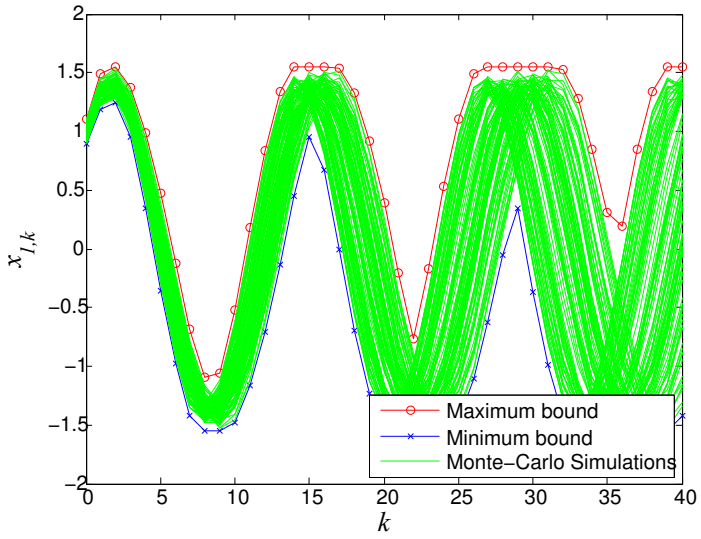

Figure 3.2: Bounds for $x_{1,k}$

3.2 Oscillatory state

The following example is a discretised rotating system [20]:

$$\begin{bmatrix} x_{1,k+1} \\ x_{2,k+1} \end{bmatrix} = \phi(x_k, p) = \frac{1}{\sqrt{1+p^2}} \begin{bmatrix} 1 & p \\ -p & 1 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} \tag{3.7}$$

In the original algorithm in [20], the uncertainty must be in a polynomial format and $\sqrt{1+p_c^2}$ was used instead of $\sqrt{1+p^2}$. Here, it does not need to be polynomial and the original form, $\sqrt{1+p^2}$, is used. The intervals for the initial state and the uncertain parameters, p, are given as follows: $0.9 \leq x_{0,1} \leq 1.1$, $0.9 \leq x_{0,2} \leq 1.1$, and $0.45 \leq p \leq 0.55$.

Algorithm 2.2.5 firstly calculates the pre upper bound for $|\phi|$, $1/\underline{\kappa}$, and set $s = 2$. Secondly, the bounds for $\max(\phi)$ and $\min(\phi)$ are obtained by Algorithms 2.2.7 and 2.2.6. For many practical systems, extreme (both maximum and the minimum) likely occurs at the boundary of at least one uncertain parameters. Hence, frequently, sweeping algorithm over r, Algorithm 2.2.9, is not required but the running algorithms for $r = 1$ is sufficient.

Figure 3.2 shows the bounds for $x_{1,k}$, where $k = 1, 2, \ldots, 39, 40$. The upper

and lower bounds of the maximum and the minimum for $r = 1$ shown in Figure 3.2 are very close to each other. All trajectories from random simulations, where the samples are from all r in $(0, 1]$, are well bounded by the estimated bounds.

The calculation time becomes longer as k increases. A simple solution is resetting the initial starting step once in a while as presented in [20]. This, however, will reduce the computational time with the cost of less tight bounds. On the other hand, the state bounds prediction in practice may not require very large k.

3.3 ErbB Signalling Pathways

Epidermal growth factor receptor (ErbB) related pathways are among the most extensively studied biological signalling networks [14]. Abnormality of ErbB signalling pathways cause various human cancers [29, 30, 31]. In [14], an ErbB mathematical model including 13 known ErbB ligands, Epidermal Growth Factor (EGF) and Heregulin (HGF) and Erk and Akt pathways are presented. It has 504 states, 828 reactions and 226 kinetic parameters. The set of 504 differential equations is extracted from the simbiology model [14]. It is known that this model is only valid up to a few hours and it is not necessary for this system to be stable for infinite time interval. As long as the states remain in a certain bound, the network works perfectly as it should be. Hence, the required robustness analysis is obtaining the information about how the future state bounds propagate with respect to the uncertain parameters.

One of the interesting biological features found in [14] using the model is that parametric sensitivities of the dynamical system strongly depend on input condition. This could be the reason that it provides so diverse responses. Parametric uncertainties are introduced for those 226 kinetic parameters. The uncertainty ranges are set to $\pm 10\%$ from the nominal values. Among the several input conditions, the robustness is tested for the case of EGF equal to 5nM. The bounds for phospholilated ErbB1, i.e. p-ErbB1, is shown in Figure 3.3. Again,

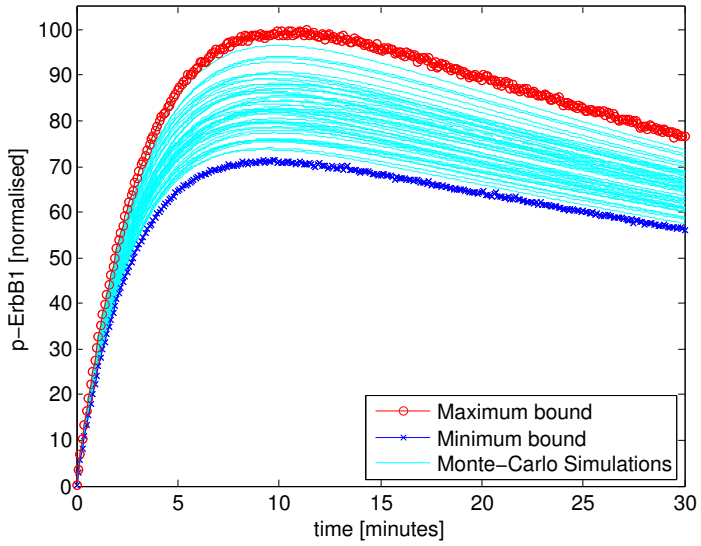

Figure 3.3: Bounds for p-ErbB1 with respect to uncertainties in 226 kinetic parameters

the upper and lower bounds for the maximum and the minimum are very close to each other and only the upper bounds for both are indicated. All trajectories from random simulations are well bounded by the estimated bounds. For the bounds calculation, r is fixed to 1 and for the random simulations, r is randomly selected between 0 and 1.

As the size of the system is relatively large, the integration part is implemented over CUDA-GPU [27] and the programmes are shown in Appendix B.

3.4 Inverted Pendulum: Hybrid System

A switching controller for inverted pendulum stabilisation is shown in [32]. A simplified version of the system is given by

$$\ddot{\theta} = p \sin\theta - u \cos\theta, \tag{3.8}$$

where θ is the angle of pendulum measured from the upright position, p is the uncertainty caused by some physical parameters, and u is the control input.

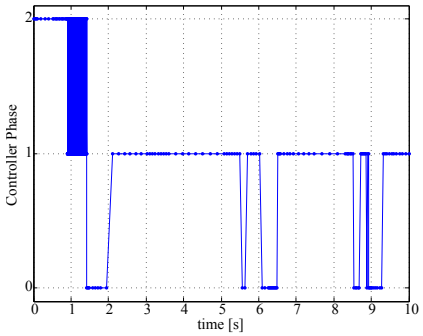

Figure 3.4: Controller switching phase examples: 0 for waiting; 1 for the energy dissipation control; and 2 for feedback linearisation control.

The total energy, E, including kinetic and potential energy is given by

$$E = \frac{1}{2}\dot{\theta}^2 + (\cos\theta - 1)$$

The controller proposed in [32] has the following switching behaviours:

- *Energy dissipation phase* (phase 1): if $|E| > \epsilon$, where ϵ is a positive real number, then

$$u = \frac{\text{sign}(E)\dot{\theta}}{1 + |\dot{\theta}|} \tag{3.9}$$

- *Waiting phase* (phase 0): if $|E| \leq \epsilon$ and $|\dot{\theta}| + |\theta| > \delta$,

$$u = 0 \tag{3.10}$$

- *Feedback linearisation control phase* (phase 2): if $|E| \leq \epsilon$ and $|\dot{\theta}| + |\theta| \leq \delta$,

$$u = \frac{2\dot{\theta} + \theta + \sin\theta}{\cos\theta} \tag{3.11}$$

The ranges for the initial values are set to: $|\theta(0)| \leq 19°$, $|\dot{\theta}(0)| \leq 20°/\text{s}$, the uncertainty range is given by $0.1 \leq p \leq 1.9$, i.e. $\pm 90\%$ uncertainty from the nominal value, 1, and ϵ and δ are set to 0.1 and 0.8, respectively. An example of the controller phase switching history is shown in Figure 3.4. The

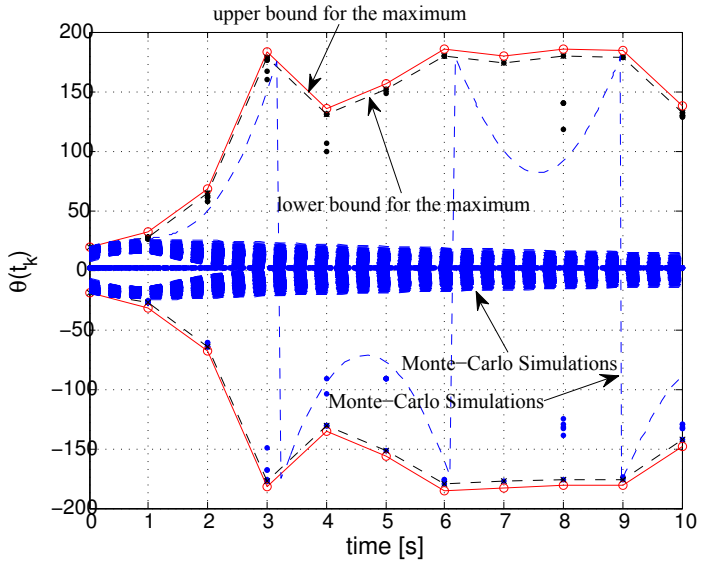

Figure 3.5: Bounds for $\theta(t_k)$

system dynamics is highly nonlinear because of its inherent nonlinearities and the switching control.

The estimated bounds obtained with $r = 1$ are shown in Figure 3.5. Although it only requires to calculate the bounds for one of either sign as the system is symmetric, for the demonstration purpose, both bounds are calculated. At $t = 10s$, the bounds are between $\pm 140°$ but most of the trajectories found by Monte-Carlo simulations converge to zero. In fact, the Monte-Carlo simulation method finds only one trajectory but still far from the bound at $t = 10$s.

This clearly demonstrates the advantage of the proposed bound algorithm over the blind Monte-Carlo simulations. The number of samples, N, for this example is set to 1024x10 and the one for Monte-Carlo simulations is twice more than N used. The calculation time of the presented algorithm for each instance is less than 0.5s. Monte-Carlo simulations takes significantly longer time, about 3 minutes, which would vary depending on the number of samples. The Monte-Carlo random simulation cannot find any solution closer to the lower bounds at $t = 10$s.

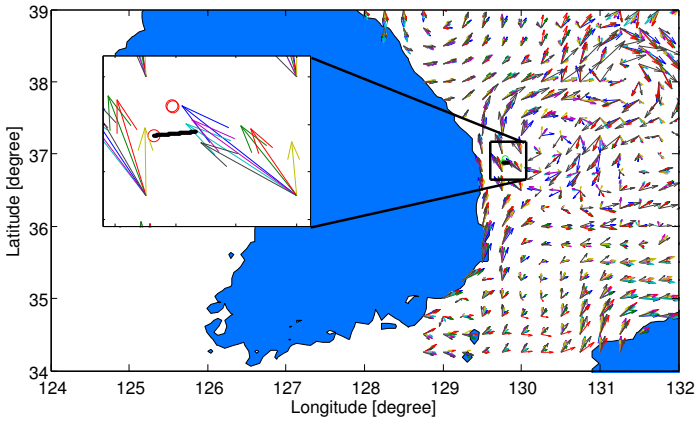

Figure 3.6: The experiment was performed in the rectangular box. The arrows are the average current velocity at each location for seven different dates from 11th to 17th March of 2011 [33]. The red circles in the inset box are the position information from GPS and the black thick line is the path obtained by the navigation system.

3.5 Underwater Glider Navigation System

In this example, Kinematics for underwater glider is derived, and the bounds for the future position of underwater glider are to be calculated using the error model including sea current and uncertainties in physical parameters. All experimental data is obtained by Korea Institute of Ocean Science & Technology (KIOST) in the area indicated in Figure 3.6.

3.5.1 Kinematics

The relation among the position vector of the underwater glider, $\mathbf{r}_b$, the sensor position in the body-coordinates ($\mathcal{B}$), $\mathbf{r}_s$, and the sensor position in the reference-coordinates ($\mathcal{R}$), $\mathbf{r}$, is shown in Figure 3.7 and

$$\mathbf{r}_b = \mathbf{r}^{\mathcal{B}} - \mathbf{r}_s^{\mathcal{B}}, \tag{3.12}$$

where $\mathbf{r}_b$ is the position vector expressed by two vectors in the right hand side, and $\mathcal{B}$ in the superscripts is indicated that both vectors are expressed in the

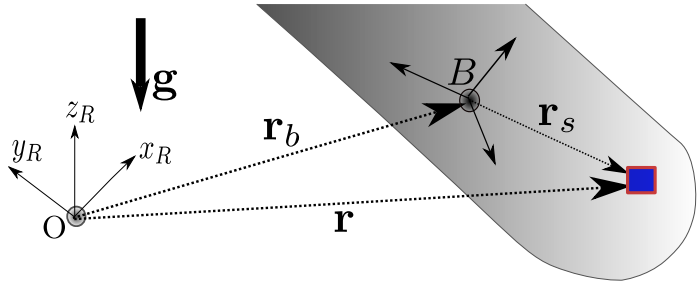

Figure 3.7: The origins of the reference-coordinates and the body-coordinates are indicated by O and B, respectively. The body coordinates is attached to the glider and $\mathbf{r}_s$ is the sensor position vector relative to the origin of the body-coordinates. $\mathbf{g}$ is the gravitational acceleration vector.

body-coordinates. Take the time-derivative of (3.12) and obtain the velocity,

$$\mathbf{v}_b = \frac{d}{dt}\mathbf{r}_b = \frac{d}{dt}\mathbf{r}^{\mathcal{B}} - \boldsymbol{\omega} \times \mathbf{r}_s^{\mathcal{B}}, \tag{3.13}$$

where $\dot{\mathbf{r}}_s^{\mathcal{B}}$ is zero because the glider is assumed to be a rigid-body, and the following transport theorem is used in the derivative [34]. Take one more time-derivative of (3.13)

$$\mathbf{a}_b = \frac{d}{dt}\mathbf{v}_b = \frac{d^2}{dt^2}\mathbf{r} - \dot{\boldsymbol{\omega}} \times \mathbf{r}_s - \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r}_s)\,, \tag{3.14}$$

where the superscript, $\mathcal{B}$, in the terms in the right hand side is dropped for brevity, and they are all expressed in the body-coordinates.

The following quantity, $\mathbf{a}_{\text{acc}}^{\mathcal{B}}$, is the measurement of the 3-axis accelerometer in the glider:

$$\mathbf{a}_{\text{acc}}^{\mathcal{B}} = \frac{d^2}{dt^2}\mathbf{r} + C_{\mathcal{R}}^{\mathcal{B}}(\psi, \theta, \phi)\mathbf{g}^{\mathcal{R}}, \tag{3.15}$$

where $C_{\mathcal{R}}^{\mathcal{B}}(\psi, \theta, \phi)$ is the direction cosine matrix, which changes the coordinates from the reference to the body with the rotating sequence given by yaw(ψ), pitch(θ), and roll(ϕ), $\mathbf{g}^{\mathcal{R}} = [0, 0, -g]^T$, the gravitational acceleration expressed

in the reference coordinates, and $g = 9.821$ m/s^2. Substitute $d^2\mathbf{r}/dt^2$ in (3.14) by the expression in (3.15)

$$\mathbf{a}_b = \frac{d}{dt}\mathbf{v}_b = \mathbf{a}_{\text{acc}}^{\mathcal{B}} - C_{\mathcal{R}}^{\mathcal{B}}\mathbf{g}^{\mathcal{R}} - \dot{\boldsymbol{\omega}} \times \mathbf{r}_s - \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r}_s)\,. \tag{3.16}$$

Rotational angular velocity of underwater vehicle is, in general, very slow, i.e. $\|\boldsymbol{\omega}\|$ and $\|\dot{\boldsymbol{\omega}}\|$ are much smaller than $\|\mathbf{g}\|$. In the experiment the magnitudes of two terms are at least 100-times smaller than the magnitude of the gravitational acceleration. Hence,

$$\mathbf{a}_b \approx \mathbf{a}_{\text{acc}}^{\mathcal{B}} - C_{\mathcal{R}}^{\mathcal{B}}\mathbf{g}^{\mathcal{R}}. \tag{3.17}$$

After transforming the coordinates of (3.17) into the reference-coordinates, integrating it with respect to time provides the velocity and the position expressed in the reference frame.

3.5.2 Error Model

Exocetus Coastal glider has a built-in navigation algorithm and it records various navigation data. The following derivations are based on the log-file obtained from the glider navigation system. The 3-Axis Acceleration measurements expressed in the body-coordinates, $\tilde{\mathbf{a}}_{\text{acc}}^{\mathcal{B}}$, are directly available from the log-file and it can be expressed as follows:

$$\tilde{\mathbf{a}}_{\text{acc}} = \tilde{\ddot{\mathbf{r}}} + \mathbf{C}_{\mathcal{R}}^{\mathcal{B}}(\psi, \theta, \phi)\mathbf{g}^{\mathcal{R}} + \mathbf{b}_{\text{acc}} + \mathbf{n}_{\text{acc}}, \tag{3.18}$$

where $\tilde{\ddot{\mathbf{r}}}$ is the measured quantity by the calibrated accelerometer if there is no stochastic noise and bias error, $\mathbf{b}_{\text{acc}}$ is the accelerometer sensor bias error, $\mathbf{n}_{\text{acc}}$ is the white noise, and the accelerations caused by the control forces and the external disturbances are all included in $\ddot{\mathbf{r}}$.

Usually, in the estimation problem, $\tilde{\ddot{\mathbf{r}}}$ is assumed to equal to $\ddot{\mathbf{r}}$, which presumes infinite resolution and infinitesimal time constant, i.e. a perfect sensitiv-

ity. In reality, including the limitations in the resolution and the finite response time, and some unknown error sources, $\tilde{\ddot{\mathbf{r}}}$ will be different from $\ddot{\mathbf{r}}$ [35].

Attitude Error: The navigation algorithm provides the attitude information, which includes some errors, $\delta\psi$, $\delta\theta$, and $\delta\phi$, then

$$C_{\mathcal{R}}^{\mathcal{B}} = C_{\mathcal{B}'}^{\mathcal{B}}(\delta\psi, \delta\theta, \delta\phi)\ C_{\mathcal{R}}^{\mathcal{B}'}(\tilde{\psi}, \tilde{\theta}, \tilde{\phi}), \tag{3.19}$$

where $\mathcal{B}'$ is the body-coordinates that the navigation system calculated, $\tilde{\psi}, \tilde{\theta}, \tilde{\phi}$ are roll, pitch and yaw angles returned from the navigation algorithm, and $\delta\psi, \delta\theta, \delta\phi$ are the attitude angles of the true body-coordinates, $\mathcal{B}$, with respect to the calculated body-coordinates, $\mathcal{B}'$.

Accelerometer Resolution Limit: The acceleration of the glider, (3.17), is calculated as follows:

$$\hat{\mathbf{a}}_b = \tilde{\mathbf{a}}_{\text{acc}}^{\mathcal{B}} - C_{\mathcal{R}}^{\mathcal{B}'}\mathbf{g}^{\mathcal{R}}, \tag{3.20}$$

Once the bias correction is completed, the position information can be obtained by integrating the corrected accelerometer output twice. The calculated path is shown in the inset of Figure 3.6 indicated in the black solid line. The starting position was initialised by the Grobal Positioning System (GPS) and it can be considered as an accurate starting position initialisation. However, because of acceleration below the sensor resolution is accumulated during the diving, the position at the end, just before updated by GPS again, has several km error after about 3 hours operation under the water. The main cause of this error might be the current as the final position is almost exactly drifted towards the directions where the average current flows. The similar drifts are observed in another three divings [36].

Sea Current Uncertainty: $\Delta\mathbf{v}$ is the relative velocity of the glider in the

horizontal plane, i.e.

$$\Delta \mathbf{v} = \begin{bmatrix} \hat{v}_{\mathrm{E}}^{\mathrm{crt}} \\ \hat{v}_{\mathrm{N}}^{\mathrm{crt}} \\ 0 \end{bmatrix} - \begin{bmatrix} v_x \\ v_y \\ 0 \end{bmatrix}, \tag{3.21}$$

$\hat{v}_E^{\mathrm{crt}}$ and $\hat{v}_{\mathrm{N}}^{\mathrm{crt}}$ are the estimated sea current velocity in the east or the north direction, respectively, whose magnitude is around 1kt (≈ 0.5 m/s) in the maximum,

$$v_{\mathrm{E}}^{\mathrm{crt}} = \hat{v}_{\mathrm{E}}^{\mathrm{crt}}(1 + \delta v_{\mathrm{E}}), \tag{3.22a}$$

$$v_{\mathrm{N}}^{\mathrm{crt}} = \hat{v}_{\mathrm{N}}^{\mathrm{crt}}(1 + \delta v_{\mathrm{N}}), \tag{3.22b}$$

$v_{\mathrm{E}}^{\mathrm{crt}}$ and $v_{\mathrm{N}}^{\mathrm{crt}}$ are the true sea current velocity in the east or the north direction, respectively, and δv_{E} and δv_{N} are the uncertainties.

Parameters Uncertainty in Drag: To quantify how much acceleration would be generated from the current, the following equation is used [37]:

$$D = \frac{1}{2}\rho \left\|\Delta \mathbf{v}\right\|^2 A_D C_D \left(1 + \delta D\right), \tag{3.23}$$

where ρ is the sea water density, which is around 1020 kg/m^3, A_D is the cross section area of the glider hull, which is about 824 cm^2, and C_D is the drag coefficient set to 0.4, which is adopted from [37]. Unlike the example shown in [37], the glider has a small wing and the induced drag is ignored. As these physical parameters are estimated using a similar size glider, the calculated drag has some error and it is represented by δD. With the nominal values, i.e. $\delta D = 0$, the drag is about 4N when $v_x = 0$, and $v_y = 0$. It corresponds to the acceleration of 0.04 m/s^2 ($\approx$ 4N/109kg). The acceleration, 0.04 m/s^2, is the possible maximum value when the glider is stationary. The acceleration caused by the current would be much smaller than the maximum as the glider usually

does not fly in the exact opposite to the current. Then, these tiny current effect would be completely blocked by the sensor noise and the external fluctuations.

3.5.3 Longitude/Latitude Bound Estimation

The acceleration caused by the drag with considering the sea current is given by

$$\mathbf{w} \approx -C_{\mathcal{B}'}^{\mathcal{R}} \Delta\ddot{\mathbf{r}} = \frac{D}{m}\mathbf{e}_D, \tag{3.24}$$

where m is the mass of the glider and $\mathbf{e}_D$ is the unit vector towards the current direction. Sea current near the surface is mainly caused by wind and the temperature differences. The vertical direction is negligible in the near surface depth compared to the magnitudes of the horizontal direction components [38]. Hence,

$$\mathbf{e}_D = \begin{cases} \dfrac{\Delta\mathbf{v}}{\|\Delta\mathbf{v}\|}, \text{ for } \|\Delta\mathbf{v}\| > 0 \\ \mathbf{0}, \text{ otherwise} \end{cases}, \tag{3.25}$$

Scenario #1: Set the uncertain parameters as follows:

$$-5^\circ \leq \delta\psi \leq 5^\circ, \tag{3.26a}$$

$$-5^\circ \leq \delta\theta \leq 5^\circ, \tag{3.26b}$$

$$-5^\circ \leq \delta\phi \leq 5^\circ, \tag{3.26c}$$

$$-0.2 \text{ [m/s]} \leq \delta v_{\mathrm{E}} \leq 0.2 \text{ [m/s]}, \tag{3.26d}$$

$$-0.2 \text{ [m/s]} \leq \delta v_{\mathrm{N}} \leq 0.2 \text{ [m/s]}, \tag{3.26e}$$

$$-0.1 \leq \delta D \leq 0.1, \tag{3.26f}$$

and the nominal values of the sea current velocity are: $\hat{v}_{\mathrm{E}}^{\mathrm{crt}} = -0.1165$ m/s and

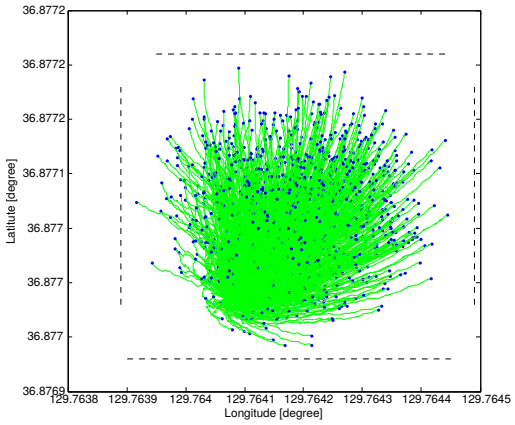

Figure 3.8: Paths after 1 minutes by Monte Carlo simulations are indicated by green lines and the lower bounds are indicated by dashed lines. The dot at the end of each path is the final location of the glider. All dots are inside the box defined by the dashed line.

$\hat{v}_{\mathrm{N}}^{\mathrm{crt}} = 0.3948$ m/s. The bounds for the possible longitude and latitude coordinates after 60s are calculated with $N = 100$ and r is fixed to 1. The four lower bounds are indicated in Figure 3.8. All paths start at the same coordinates and the final position for each random simulation is indicated by blue dot. All final positions are well bounded by the lower bound. The upper bounds are not indicated.

Scenario #2: As the uncertain ranges in Scenario #1 are too big for longer time horizon, i.e., the paths will spread broad region with longer final time, reduce the uncertain parameter ranges by 10 as follows:

$$-0.5^\circ \leq \delta\psi \leq 0.5^\circ, \tag{3.27a}$$

$$-0.5^\circ \leq \delta\theta \leq 0.5^\circ, \tag{3.27b}$$

$$-0.5^\circ \leq \delta\phi \leq 0.5^\circ, \tag{3.27c}$$

$$-0.02 \text{ [m/s]} \leq \delta v_{\mathrm{E}} \leq 0.02 \text{ [m/s]}, \tag{3.27d}$$

$$-0.02 \text{ [m/s]} \leq \delta v_{\mathrm{N}} \leq 0.02 \text{ [m/s]}, \tag{3.27e}$$

$$-0.01 \leq \delta D \leq 0.01. \tag{3.27f}$$

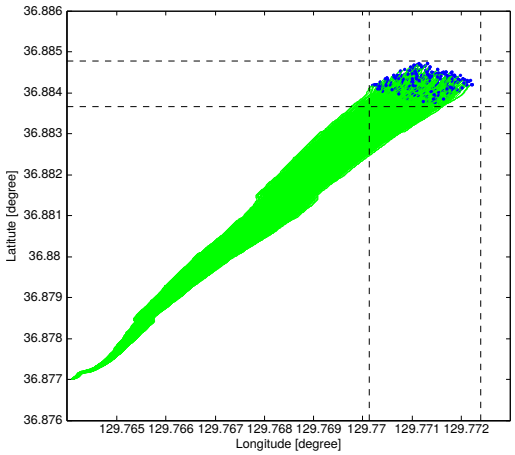

Figure 3.9: Paths after 1 hour by Monte Carlo simulations are indicated by green lines and the lower bounds are indicated by dashed lines. The dot at the end of each path is the final location of the glider. All dots are inside the box defined by the dashed line.

The final time is set to 1 hour. The lower bounds shown in Figure 3.9 confine tightly all the possible positions after 1 hour. The number of Monte Carlo simulation is 500.

Chapter 4

Conclusions

Algorithms for calculating state bounds for general nonlinear systems with uncertain initial states and parameters are developed. The algorithms combine μ-formulation for optimisation problem and pseudo-LFT for LFT-free formulation. The algorithms have several advantages including: 1) no effort to obtain an LFT form; 2) easy to parallelise on distributed computers; and 3) applicable to many types of functions including the one having finite number of discontinuity. The algorithms are applied to multi-dimensional Rosenbrock function; a simple oscillatory nonlinear discrete system; a high-dimensional biological model for ErbB signalling pathways; a hybrid system with discontinuity; and finally, navigation error propagation of underwater glider.

It is highly desirable to have numerically efficient algorithms to estimate state bounds for general nonlinear systems. Especially, the input-output robustness analysis with respect to various parametric perturbations are one of the main interest in the robustness of biological networks; autonomous mobile robots operating in uncertain environment requires to predict the future state bounds in order to plan or re-plan its behaviour; predicting a group of space debris is very important for the safety of space mission. The suggested algorithms could be the main tool to analyse such complex nonlinear systems with uncertainties.

As demonstrated by the multi-dimensional Rosenbrock functions, however,

the gap between the lower bounds and the upper bounds increases, in general, as the search dimension increases. This is the inherent complexity of NP-hard problem, which cannot be overcome unless NP=P. On the other hand, up to certain dimensions, tighter gap can be obtained with much shorter computation time by parallelising the algorithms. We demonstrated the possibility of parallel processing in real-time using GPU. In future, it will be possible to perform real-time optimisation for predicting uncertain bounds and adjusting control gains to reduce the bounds with fast massive parallel processing computation.

Finally, it is worth to note that a similar algorithm to the proposed ones was presented in [39], which is called Luus-Jaakola algorithm, and its convergence property was shown in [40]. It uses random sampling and re-sampling based on the values from the random samples. It was presented in 1973 and its usages were limited to small size problems because of a lot restricted computational power in the past. The re-sampling method could be adopted in the proposed algorithm to reduce the computational cost.

References

[1] Gary J. Balas, John C. Doyle, Keith Glover, Andy Packard, and Roy Smith. *μ-Analysis and Synthesis Toolbox: For Use with MATLAB, User's Guide, Version 3*. The MathWorks, Inc., 2001.

[2] Gilles Ferreres and Jean-Marc Biannic. Reliable computation of the robustness margin for a flexible aircraft. *Control Engineering Practice*, 9:1267–1278, 2001.

[3] Prathyush P. Menon, Jongrae Kim, Declan G. Bates, and Ian Postlethwaite. Clearance of nonlinear flight control laws using hybrid evolutionary optimisation. *IEEE Transactions on Evolutionary Computation*, 10(6):689–699, December 2006.

[4] Hiroaki Kitano. Biological robustness. *Nat Rev Genet*, 5(11):826–837, November 2004.

[5] Andreas Wagner. Circuit topology and the evolution of robustness in two-gene circadian oscillators. *Proceedings of the National Academy of Sciences of the United States of America*, 102(33):11775–11780, August 2005. PMID: 16087882.

[6] Mark Gilchrist, Vesteinn Thorsson, Bin Li, Alistair G. Rust, Martin Korb, Kathleen Kennedy, Tsonwin Hai, Hamid Bolouri, and Alan Aderem. Systems biology approaches identify ATF3 as a negative regulator of toll-like recepter 4. *Nature*, 441(11):173–178, May 2006.

[7] Jongrae Kim, Declan G. Bates, Ian Postlethwaite, Lan Ma, and Pablo Iglesias. Robustness analysis of biochemical networks models. *IEE Systems Biology*, 153(3):96–104, May 2006.

[8] Guy Shinar, Ron Milo, Mara Rodrguez Martnez, and Uri Alon. Input output robustness in simple bacterial signaling systems. *Proceedings of the National Academy of Sciences of the United States of America*, 104(50):19931–19935, December 2007. PMID: 18077424.

[9] Sbastien Clodong, Ulf Dhring, Luiza Kronk, Annegret Wilde, Ilka Axmann, Hanspeter Herzel, and Markus Kollmann. Functioning and robustness of a bacterial circadian clock. *Molecular Systems Biology*, 3:90, 2007. PMID: 17353932.

[10] Murat Acar, Jerome T Mettetal, and Alexander van Oudenaarden. Stochastic switching as a survival strategy in fluctuating environments. *Nature Genetics*, 40(4):471–475, April 2008. PMID: 18362885.

[11] Boris N. Kholodenko, Anatoly Kiyatkin, Frank J. Bruggeman, Eduardo Sontag, and Hans V. Westerhoff. Untangling the wires: A strategy to trace functional interactions in signaling and gene networks. *Proceedings of the National Academy of Sciences*, 99(20):12841–12846, October 2002.

[12] Robert Hoehndorf, Michel Dumontier, John H. Gennari, Sarala Wimalaratne, Bernard de Bono, Daniel L. Cook, and Georgios V. Gkoutos. Integrating systems biology models and biomedical ontologies. *BMC systems biology*, 5(1):124+, August 2011.

[13] Clemens Kuhn, Christoph Wierling, Alexander Kuhn, Edda Klipp, Georgia Panopoulou, Hans Lehrach, and Albert Poustka. Monte Carlo analysis of an ODE Model of the Sea Urchin Endomesoderm Network. *BMC Systems Biology*, 3(1):83+, 2009.

[14] William W. Chen, Birgit Schoeberl, Paul J. Jasper, Mario Niepel, Ulrik B. Nielsen, Douglas A. Lauffenburger, and Peter K. Sorger. Input-output

behavior of ErbB signaling pathways as revealed by a mass action model trained against dynamic data. *Molecular Systems Biology*, 5, January 2009.

[15] John Doyle. Analysis of feedback systems with structured uncertainties. *IEEE Transactions on Aerospace and Electronic Systems*, 129(6):242–250, November 1982.

[16] S. Skogestad and I. Postlethwaite. *Multivariable Feedback Control: Analysis and Design*. John Wiley & Sons Ltd., 1996.

[17] Chung-Yao Kao, Alexandre Megretski, and Ulf T. Jönsson. A cutting plane algorithm for robustness analysis of periodically time-varying system. *IEEE Transactions on Automatic Control*, 46(4):579–592, April 2001.

[18] M. Cantoni and K. Glover. Gap-metric robustness analysis of linear periodically time-varying feedback systems. *SIAM Journal on Control and Optimization*, 38(3):803–822, 2000.

[19] R.P. Braatz, P.M. Young, J.C. Doyle, and M. Morari. Computational complexity of μ calculation. *Automatic Control, IEEE Transactions on*, 39(5):1000 –1002, May 1994.

[20] Masako Kishida, Philipp Rumschinski, Rolf Findeisen, and Richard D. Braatz. Efficient polynomial-time outer bounds on state trajectories for uncertain polynomial systems using skewed structured singular values. In *IEEE Multi-Conferences on Systems and Control*, Denver, CO., USA, 27 - 30 September 2011.

[21] John C. Doyle, Keith Glover, Pramod P. Khargonekar, and Bruce A. Francis. State-space solutions to standard h_2 and h_∞ control problems. *IEEE Transactions on Automatic Control*, 34(8):831–847, August 1989.

[22] J. Kim, D.G. Bates, and I. Postlethwaite. A geometrical formulation of the mu-lower bound problem. *IET Control Theory & Applications*, 3(4):465–472, 2009.

[23] Yun-Bo Zhao, Jongrae Kim, and Declan G. Bates. LFT-free robustness analysis of LTI/LPTV systems. In *IEEE Multi-Conferences on Systems and Control*, Denver, CO., USA, 27 - 30 September 2011.

[24] Jongrae Kim and Kevin Worrall. Sun tracking controller using for UKube-1 using magnetic torquer only. In *19th IFAC Symposium on Automatic Control in Aerospace*, Wuerzburg, Germany, 2013.

[25] Tae W. Lim. Thruster Attitude Control System Design and Performance for Tactical Satellite 4 Maneuvers. *Journal of Guidance, Control, and Dynamics*, 37(2):403–412, February 2014.

[26] Concepción A. Monje, Blas M. Vinagre, Vicente Feliu, and YangQuan Chen. Tuning and auto-tuning of fractional order controllers for industry applications. *Control Engineering Practice*, 16(7):798–812, July 2008.

[27] NVIDIA Developer zone: http://developer.nvidia.com, April 2014.

[28] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):493–507, December 1952.

[29] Jeffrey A. Engelman *et. al.* Met amplification leads to gefitinib resistance in lung cancer by activating erbb3 signaling. *Science*, 316(5827):1039–1043, 2007.

[30] Wenjun Zhou *et. al.* Novel mutant-selective EGFR kinase inhibitors against EGFR T790M. *Nature*, 462(7276):1070–1074, December 2009.

[31] Kimio Yonesaka *et. al.* Activation of erbb2 signaling causes resistance to the egfr-directed therapeutic antibody cetuximab. *Science Translational Medicine*, 3(99):99ra86, 2011.

[32] K. J. Åström and K. Furuta. Swinging up a pendulum by energy control. *Automatica*, 36(2):287–295, February 2000.

[33] Korea Hydrographic & Oceanographic Administration. East sea surface currents prediction model. 2011.

[34] H. Schaub and J. Junkins. *Analytical Mechanics of Space Systems (AIAA Education Series)*. AIAA, 2 edition, October 2009.

[35] A. V. Ulanov. Underwater glider navigation in a nonhomogenous current. *Aerospace and Electronic Systems Magazine, IEEE*, 21(11):27–28, 2006.

[36] LLC. Alaks Native Technologies. ANT analysis of KORDI LG19 flight. Technical Memo, 2011.

[37] J. Sherman, R. Davis, W. B. Owens, and J. Valdes. The autonomous underwater glider "Spray". *Oceanic Engineering, IEEE Journal of*, 26(4):437–446, October 2001.

[38] Young-Gyu Park, Jae-Hun Park, Ho J. Lee, Hong S. Min, and Seon-Dong Kim. The effects of geothermal heating on the East/Japan Sea circulation. *J. Geophys. Res. Oceans*, 118(4):1893–1905, April 2013.

[39] Rein Luus and T. H. I. Jaakola. Optimization by direct search and systematic reduction of the size of search region. *AIChE J.*, 19(4):760–766, July 1973.

[40] G. Gopalakrishnan Nair. On the convergence of the LJ search method. 28(3):429–434, 1979.

List of Symbols and Abbreviations

A_D	underwater glider cross section area	34
$\mathcal{B}$	body frame	30
$C_{\mathcal{R}}^{\mathcal{B}}$	direction cosine matrix from $\mathcal{R}$ to $\mathcal{B}$	31
C_D	underwater glider drag coefficient	34
D	drag	34
E	largest positive number that the computer can express	16
N	total number of samples	16
N_r	total number of samples in $(0, 1)$	18
R	search radius set	18
$\mathcal{R}$	reference frame	30
$\mathcal{V}_i$	connected sub-volume in the search space	20
$\mathcal{V}_T$	total volume of the search space	20
$\mathbf{a}_b$	underwater glider acceleration	31
$\mathbf{b}_{\text{acc}}$	accelerometer bias error	32
$\mathbf{e}_D$	unit vector towards the drag	35
$\mathbf{g}$	gravitational acceleration	31
m	glider mass	35
$\mathbf{n}_{\text{acc}}$	accelerometer white noise	32

n_l number of samples, whose values are greater than the current lower bound 20

n_x state vector dimension 7

p uncertain parameters in the rotating system or the hybrid system 25

p_i uncertain parameters in the Rosenbrock function 22

$p(r)$ sum of all sub-volumes, where values greater than bound, to the total volume . 19

$\hat{p}(r)$ estimated $p(r)$ 20

r search radius in ∞-norm, which is between 0 and 1 14

$\tilde{\ddot{\mathbf{r}}}$ calibrated accelerometer measurement 32

$\mathbf{r}_b$ underwater glider position with respect to the reference frame 30

$\mathbf{r}_s$ underwater glider sensor position with respect to the body frame 30

r_s sampled search radius in ∞-norm 18

s shifting factor greater than 1 17

s_f safety factor greater than 1 17

t time 7

u control input in the hybrid system 27

$\mathbf{v}_b$ underwater glider velocity 31

$\hat{v}_{\mathrm{E}}^{\mathrm{crt}}$ sea current velocity in East direction 33

$\hat{v}_{\mathrm{N}}^{\mathrm{crt}}$ sea current velocity in North direction 33

$\mathbf{w}$ acceleration caused by the sea current 35

x state vector 7

x_i^k i-th state at k-th sampling time of the rotating system 25

δ_c confidence level in Chernoff inequality 19

δD uncertainty in the drag 35

δv_{E} sea current velocity uncertainty in East direction 34

δv_{N} sea current velocity uncertainty in North direction 34

$\delta\psi$ roll angle estimation error 35

$\delta\theta$ pitch angle estimation error 35

$\delta\phi$ yaw angle estimation error 35

ϵ smallest positive number that the computer can express 16

ϵ_c error bound in Chernoff inequality 19

ε κ bounds tolerance, $E - \epsilon$ 16

θ angular state in the hybrid system 27

κ inverse of the absolute maximum of the function to be maximised, $E - \epsilon$ 14

$\bar{\kappa}$ upper bound of κ^* 15

$\underline{\kappa}$ lower bound of κ^* 15

κ^* true κ 14

$\underline{\kappa}_p$ pre-κ bound 16

ρ sea water density, 1020 kg/m^3 34

$\boldsymbol{\omega}$ underwater glider angular velocity 31

List of Acronyms

EGF Epidermal Growth Factor

ErbB Epidermal growth factor receptor

GPS Grobal Positioning System

GPU Graphical Processing Unit

HGF Heregulin

KIOST Korea Institute of Ocean Science & Technology

LFT Linear Fractional Transformation

NP-hard Non-deterministic Polynomial-time hard

Appendices

Appendix A

Hello World example

A.1 main_get_max_all.m

```
%-------------------------------------------------------
% Main: States bound calculation for many k steps
%-------------------------------------------------------
%
% See #1, #2 and #3 below

% Programmed by Jongrae <Jongrae.Kim@glasgow.ac.uk>
% University of Glasgow
% Biomedical Engineering
% RobustLab: http://www.robustlab.org

% Change Log:
% 1. 15th January 2014
%   To solve optimisation problem
% 2. 15th Feburuary 2014
%   To correct algorithm for finding solutions inside boundary
% 3. 1st April 2014
%   To find the maximum or the minimum; remove the positiveness condition

clear all;

%%
%-------------------------------------------------------
% 1. Algorithm Parameters that you may want to adjust
%-------------------------------------------------------

% number of samples in one Period
num_steps = 1;
num_delta_sample = 1000;
       % lower with less computation but higher risk to be failed
       % higher with less risk but more computational burden

safety_factor_max = 1+1e-6;
safety_factor_min = 1+1e-6;
       % always greater than or equal to 1
       % smaller would give tighter upper bounds with higher risk to be failed,
       % larger would have less possbility to be failed but less tight upper bounds

% find minimum or maximum
```

```
find_min = true; % true for minimum; false for maximum

% Range of k, where k for k*delta_c
box_init = [1e-12 1e12];%[-0.5 1]*1e3;
        % For box_init = [1e-12 1e12];
        % you should know roughy where the k_optimal would be.
        % i.e, here I am quite sure that  1e-2 < f < 1e12
        % if your calculation failed, then you may adjust this

% Use adaptive box & samplings
use_adaptive_sampling = false;
num_replace = floor(num_delta_sample*0.8+0.5);

% ------------------------------------------
% 2. the function f(x,p) related setting
% ------------------------------------------
% parameter bounds
x_dim = 2;
xL = -pi*ones(x_dim,1);
xU =  sqrt(2)*pi*ones(x_dim,1);

s_scale = 1.2;

% uncertain parameters dimension
state_dim = length(xL);

w_x = (xU-xL)/2;
x_centre = xL+w_x;

w_x_prev = w_x;
x_centre_prev = x_centre;

w_x_org = w_x;
x_centre_org = x_centre;

%%
%==================================================
% No need to change from here
%==================================================
% find maximum of |f|
tic
c_max = 0;
[x_upper_max, x_lower_max, sol_opt, weight_opt]  =  ...
    multi_cal_state_bounds([], ...
    box_init,num_steps,num_delta_sample,[],state_dim, ...
    [],[],[],x_centre,w_x,[], ...
    [], [], num_replace,[],c_max, find_min);
opt_x = x_centre + weight_opt*sol_opt(:).*w_x(:);
fprintf('Optimal Sol: x = (');
for idx=1:state_dim
   fprintf('%4.2f ',opt_x(idx));
end
fprintf(')\n');

x_opt = x_centre(:)' + (sol_opt(:).*w_x(:))'*weight_opt;
opt_cost = get_f_delta(sol_opt,x_centre,w_x*weight_opt,0);
fprintf('Optimal Cost = %10.5f\n',opt_cost-c_max);
fprintf('Cost bound = [%10.5f, %10.5f]\n', ...
    x_lower_max-c_max,x_upper_max-c_max);

cal_time = toc;
fprintf('----------------------------------------\n');
```

```
fprintf('Calculation Time = %6.3f [s]\n',cal_time);
fprintf('----------------------------------------\n');

f_abs_max = opt_cost;

% find maximum of |f+f_abs_max*2|
tic
c_max = f_abs_max*s_scale;
[x_upper_max, x_lower_max, sol_opt, weight_opt]  =  ...
    multi_cal_state_bounds([], ...
    box_init,num_steps,num_delta_sample,[],state_dim, ...
    [],[],[],x_centre,w_x,[], ...
    [], [], num_replace,[],c_max, find_min);
opt_x = x_centre + weight_opt*sol_opt(:).*w_x(:);
fprintf('Optimal Sol: x = (');
for idx=1:state_dim
   fprintf('%4.2f ',opt_x(idx));
end
fprintf(')\n');

opt_cost = get_f_delta(sol_opt,x_centre,w_x*weight_opt,0);
fprintf('Optimal Cost = %10.5f\n',opt_cost);
fprintf('Cost bound = [%10.5f, %10.5f]\n',x_lower_max-c_max, ...
 safety_factor_max*x_upper_max-c_max);

cal_time = toc;
fprintf('----------------------------------------\n');
fprintf('Calculation Time = %6.3f [s]\n',cal_time);
fprintf('----------------------------------------\n');

% find minimum of |f-c_max*2|
tic
c_max = -f_abs_max*s_scale;
[x_upper_max, x_lower_max, sol_opt, weight_opt]  =  multi_cal_state_bounds([], ...
    box_init,num_steps,num_delta_sample,[],state_dim, ...
    [],[],[],x_centre,w_x,[], ...
    [], [], num_replace,[],c_max, find_min);
opt_x = x_centre + weight_opt*sol_opt(:).*w_x(:);
fprintf('Optimal Sol: x = (');
for idx=1:state_dim
   fprintf('%4.2f ',opt_x(idx));
end
fprintf(')\n');

opt_cost = get_f_delta(sol_opt,x_centre,w_x*weight_opt,0);
fprintf('Optimal Cost = %10.5f\n',opt_cost);
fprintf('Cost bound = [%10.5f, %10.5f]\n',x_lower_max+c_max, ...
    safety_factor_min*x_upper_max+c_max);

cal_time = toc;
fprintf('----------------------------------------\n');
fprintf('Calculation Time = %6.3f [s]\n',cal_time);
fprintf('----------------------------------------\n');
```

A.2 multi_cal_state_bounds.m

```
function [x_upper_max_global, x_lower_max_global, extreme_delta, extreme_w] = ...
    multi_cal_state_bounds(k_step, box_init, ...
    num_steps, num_delta_sample,delta_dim,state_dim,which_x, ...
    p_centre,w_p,x_centre,w_x,safety_factor, ...
    f_real_prev, delta_px_prev, num_replace, ...
    reset_steps, c_max, find_min)

% multiple box size
Nr = 21;
w_box_size = linspace(0,1,Nr);
max_iter   = 10000;
min_iter   = 10;
w_box_min_size = 1e-6;

w_x = w_x(:)';
[zero_cost, ] = get_f_delta(zeros(size(w_x)),x_centre,w_x,c_max);

%% calcualte upper and lower bounds for the maximum
not_converge_sw    = true;
x_upper_max_global = -inf;
x_lower_max_global =  zero_cost;
extreme_delta      = zeros(size(w_x(:)'));
extreme_w          = 0;

current_iter = 1;
prev_opt_x = zeros(size(w_x));
count_converge = 0;
w_prev_box = [];

while not_converge_sw

    sol_opt_all      = zeros(length(w_box_size),state_dim);
    opt_cost_all     = zeros(length(w_box_size),1);
    opt_cost_all(1)  = zero_cost;

    current_w_range = w_box_size(end) - w_box_size(1);

    for wdx = 2:length(w_box_size)

        w_x_current = w_x*w_box_size(wdx);

        [x_upper_max, x_lower_max, sol_opt]  = cal_state_bounds([], ...
            box_init,num_steps,num_delta_sample,[],state_dim, ...
            [],[],[],x_centre,w_x_current,[], ...
            [], [], num_replace,[],c_max, find_min);

        opt_cost = 0;
        if ~isempty(sol_opt)
            [opt_cost, ] = get_f_delta(sol_opt,x_centre,w_x_current,c_max);
        end

        if x_upper_max > x_upper_max_global
            x_upper_max_global = x_upper_max;
        end

        if x_lower_max > x_lower_max_global
            x_lower_max_global  = x_lower_max;
            extreme_delta       = sol_opt(:)';
```

```
59              extreme_w           = w_box_size(wdx);
60          end
61
62          sol_opt_all(wdx,:) = sol_opt(:)';
63          opt_cost_all(wdx)  = opt_cost;
64
65      end
66      cost_diff_sgn = sign(diff(opt_cost_all));
67      nbhd_idx = [2:length(w_box_size)] + cost_diff_sgn(:)';
68      if nbhd_idx(end)>length(w_box_size)
69          nbhd_idx(end) = length(w_box_size);
70      end
71
72      figure(1); clf;
73      [aa,bb] = sort(w_box_size);
74      plot(aa,opt_cost_all(bb),'o-');
75      hold on;
76      plot(extreme_w, x_lower_max_global, 'r^');
77      pause(0.01);
78
79      dw = (w_box_size(nbhd_idx)-w_box_size(2:end))/2;
80
81      if dw(1) > 0
82          zero_cost     = opt_cost_all(2);
83          w_box_size(1) = w_box_size(2);
84      end
85
86      dw = (w_box_size(nbhd_idx)-w_box_size(2:end)); dw = dw.*rand(size(dw));
87
88      w_box_size = w_box_size + [0 dw(:)'];
89      w_box_size = sort(w_box_size);
90
91      if w_box_size(end) > 1
92          w_box_size(end) = 1;
93      end
94
95      [aa,bb]=min(abs(w_box_size-extreme_w));
96      w_box_size(bb) = extreme_w;
97
98      %------------
99      if ~isempty(w_prev_box)
100         if length(w_box_size)==length(w_prev_box)
101             if norm(w_box_size-w_prev_box)==0
102                 not_converge_sw = false;
103             end
104         end
105     end
106     if length(w_box_size)==1
107         not_converge_sw = false;
108     end
109     w_prev_box = w_box_size;
110     %------------
111
112     new_w_range = abs(w_box_size(end)-w_box_size(1));
113     if abs(new_w_range - current_w_range) < w_box_min_size
114         count_converge = count_converge + 1;
115     else
116         count_converge = 0;
117     end
118
119     if new_w_range < w_box_min_size || current_iter > max_iter
120         not_converge_sw = false;
```

```
121     end
122
123     if count_converge > min_iter
124         w_box_size = extreme_w + extreme_w*randn(size(w_box_size))*new_w_range/10;
125         w_box_size = sort(w_box_size);
126     end
127
128     %-----------------------------
129     % intermediate output
130     %-----------------------------
131     opt_x = x_centre + extreme_w*extreme_delta(:).*w_x(:);
132     fprintf('%d-th iteration: Current Optimal Sol: x = (',current_iter);
133     for idx=1:state_dim
134         fprintf('%6.4f ',opt_x(idx));
135     end
136     fprintf(');');
137     fprintf('dw = %6.4f < %6.4f ?\n', abs(w_box_size(end)-w_box_size(1)), w_box_min_size);
138     %-----------------------------
139
140     current_iter = current_iter + 1;
141
142 end
```

A.3 cal_state_bounds.m

```
%-------------------------------------------------------------------------
% States bound calculation for discrete nonlinear systems using mu-bounds
%-------------------------------------------------------------------------
function [x_upper, x_lower, extreme_delta] = cal_state_bounds(k_step, box_init, ...
    num_steps, num_delta_sample,delta_dim,state_dim,which_x, ...
    p_centre,w_p,x_centre,w_x,safety_factor, ...
    f_real_prev, delta_px_prev, num_replace, ...
    reset_steps, c_max, find_min)
% Input:
%   k_step: bounds for k_step value of f(x,d)
%
%   box_init: initial search range of k,  k is searched in the range of
%       box_init(1) <= k <= box_init(2)
%
%   num_delta_sample: number of random samples along the 1-norm box edge
%
%   delta_dim: how many uncertain parameters
%
%   state_dim: dimension of state vector x
%
%   which_x: which x's bounds to be calculated, range = [1 state_dim]
%
%   p_centre: vector for the centres of uncertain parameters
%
%   w_p: weighting vector for the uncertain parameters so that p_delta is
%       between -1 and 1
%
%   x_centre: vector for the centres of state vector
%
%   w_x: weighting vector for the state vector so that x_delta is
%       between -1 and 1
%
%   safety_factor: safety factor for the upper bound, greater than 1,
%       smaller value may give more tighter bound but with higher risk
%       that may be wrong, this only applies to the upper bound
%
%   c_min_max: adjusting value for obtain bounds for max(f) and min(f)
%
% Output:
%   x_upper: upper bound
%   x_lower: lower bound
%   f_min_max: minimum and maximum values of f found during the sampling

% Programmed by Jongrae <Jongrae.Kim@glasgow.ac.uk>
% University of Glasgow
% Biomedical Engineering
% RobustLab: http://www.robustlab.org

%%
tol = 1e-3;

%%
current_big_box = box_init(2);
current_small_box = box_init(1);
f_min_max = [inf -inf];
current_delta_px_worst = inf;
x_lower = -inf;

```

```
if isempty(delta_dim)
    delta_dim = 0;
end

feasible_set_sw = true;

while isinf(current_delta_px_worst(1))

    init_box_saved = [current_small_box current_big_box];

    while abs(current_big_box-current_small_box) > tol

        current_box_size = (current_big_box + current_small_box)/2;
        current_k = current_box_size;

        % uniform sample in the 1-norm box random face
        delta_x = 2*rand(num_delta_sample,delta_dim+state_dim,1)-1;
        rand_face = randi(delta_dim+state_dim,num_delta_sample,1);
        idx_face = sub2ind([num_delta_sample delta_dim+state_dim], ...
                (1:num_delta_sample)',rand_face);
        delta_x(idx_face) = sign(rand(num_delta_sample,1)-0.5);

        k_delta_c = current_k;
        [f_real_org, f_delta] = I_ND(delta_x, k_delta_c, x_centre, w_x, c_max, find_min);
        [f_real, idx_f_real_min] = min(f_real_org);

        if f_real < 0

            current_delta_px_worst = delta_x(idx_f_real_min,:);

            current_big_box = current_box_size;

            cand_k = 1/(abs(f_delta(idx_f_real_min,:)));
            if cand_k < current_k
                current_k = cand_k;
                x_lower_cand = 1/cand_k;
            else
                x_lower_cand = 1/current_k;
                current_k_upper = current_k;
            end

            if x_lower_cand > x_lower
                x_lower = x_lower_cand;
                current_k_upper = current_k;
                extreme_delta = delta_x(idx_f_real_min,:);
            end
        end

        if f_real > 0
            current_small_box = current_box_size;
        end

        % no feasible set
        if sum(abs(f_delta))==0 && length(f_delta) > 1
            x_upper = 0;
            x_lower = 0;
            extreme_delta =  delta_x(1,:); % arbitrary delta for flat cost function
            feasible_set_sw = false;
            break;
        end

    end % of while: check if the boxes are close enough?
```

```
121
122     %%
123
124     % if never find f_real < 0, then adjust the initial box sizes
125     current_big_box   = init_box_saved(2)*2;
126     current_small_box = max([1/pi init_box_saved(1)/2]);
127
128     if ~feasible_set_sw
129         break;
130     end
131
132 end % of current_delta_px_worst == inf?
133
134 %%
135 % Refining the x_upper, i.e., k_lower
136 if feasible_set_sw
137
138     upper_bd_not_converge = true;
139
140     epsilon_box = tol*10;
141     k_box = [current_k_upper/10 current_k_upper];
142
143     while upper_bd_not_converge
144
145         for idx=1:num_steps
146
147             % sample in the epsilon-norm box random face
148             delta_x = 2*rand(num_delta_sample*10,delta_dim+state_dim,1)-1;
149             delta_x = epsilon_box*delta_x;
150             delta_x = kron(ones(num_delta_sample*10,1),current_delta_px_worst) + delta_x;
151
152             k_delta_c = k_box(1) + abs(diff(k_box))/2;
153
154             [f_real_org, f_delta] = I_ND(delta_x, k_delta_c, x_centre, ...
155             w_x, c_max, find_min);
156
157             f_real = min(f_real_org);
158             f_delta_min = min(f_delta,[],1);
159             f_delta_max = max(f_delta,[],1);
160
161             if f_delta_min < f_min_max(1)
162                 f_min_max(1) = f_delta_min;
163             end
164             if f_delta_max > f_min_max(2)
165                 f_min_max(2) = f_delta_max;
166             end
167
168             if f_real < 0
169                 k_box(2) = k_delta_c;
170                 break;
171             end
172
173         end
174
175         if ((idx==num_steps) && (f_real > 0))
176             upper_bd_not_converge = false;
177             x_upper = 1/k_box(2);
178         end
179
180     end % of while
181
182 end % of if ~feasible_set_sw
```

A.4 I_ND.m

```
function [f_real, f_delta] = I_ND(delta_x, k_delta_c, x_centre, w_x, c_max, find_min)
% Input:
%   k_step: bounds for k_step value of f(x,d)
%
%   delta_px: vector including delta_p and delta_x
%       e.g. delta_px = [ delta_p(1); delta_p(2); delta_x1; delta_x2]
%
%   k_delta_c: current k*delta_c
%
%   which_x: which x's bounds to be calculated, range = [1 state_dim]
%
%   p_centre: vector for the centres of uncertain parameters
%
%   w_p: weighting vector for the uncertain parameters so that p_delta is
%       between -1 and 1
%
%   x_centre: vector for the centres of state vector
%
%   w_x: weighting vector for the state vector so that x_delta is
%       between -1 and 1
%
%   c_min_max: adjusting value for obtain bounds for max(f) and min(f)
%
% Output:
%   f_real: real part of det(I-N Delta)
%   f_delta: f evaluated at the sampled point

% Programmed by Jongrae <Jongrae.Kim@glasgow.ac.uk>
% University of Glasgow
% Biomedical Engineering
% RobustLab: http://www.robustlab.org

f_delta = get_f_delta(delta_x,x_centre, w_x, c_max);
f_real = 1 - k_delta_c*abs(f_delta);

end
```

A.5 get_f_delta.m

```
function f_delta = get_f_delta(delta_x, x_centre, w_x, c_max)
% Input:
%   k_step: bounds for k_step value of f(x,d)
%
%   delta_px: vector including delta_p and delta_x
%       e.g. delta_px = [ delta_p(1); delta_p(2); delta_x1; delta_x2]
%
%   p_centre: vector for the centres of uncertain parameters
%
%   w_p: weighting vector for the uncertain parameters so that p_delta is
%       between -1 and 1
%
%   x_centre: vector for the centres of state vector
%
%   w_x: weighting vector for the state vector so that x_delta is
%       between -1 and 1
%
% Output:
%   f_delta: function f evaluated at the sampled x and p

% Programmed by Jongrae <Jongrae.Kim@glasgow.ac.uk>
% University of Glasgow
% Biomedical Engineering
% RobustLab: http://www.robustlab.org

%% DO NOT CHANGE HERE

% pertubed states
n_delta = size(delta_x,1);
delta_dim = size(delta_x,2);

x_centre = x_centre(:)';
w_x      = w_x(:)';
x_delta = kron(ones(n_delta,1),x_centre) +kron(ones(n_delta,1),w_x).*delta_x;

%% ONLY CHANGE BELOW
%-------------------------------------------------------
% f(x,p)
%-------------------------------------------------------
f_delta = zeros(size(x_delta,1),1);

for idx=1:1:delta_dim-1
    x1 = x_delta(:,idx);
    x2 = x_delta(:,idx+1);
    f_delta = f_delta + 100*(x2-x1.^2).^2 + (1-x1).^2;
end

f_delta = abs(f_delta + c_max);
%-------------------------------------------------------

end
```

Appendix B

ErbB Signalling Pathways: GPU Example

B.1 state_bounds_for_many_steps_main.m

```
%-----------------------------------------------------
% Main: States bound calculation for many k steps
%-----------------------------------------------------
%
% See #1, #2 and #3 below

% Programmed by Jongrae <Jongrae.Kim@glasgow.ac.uk>
% University of Glasgow
% Biomedical Engineering
% RobustLab: http://www.robustlab.org

clear all;

global function_handle;

num_data = 1024;%*10;
function_handle= ...
 parallel.gpu.CUDAKernel('get_f_delta_ptx_vector_form.ptx','get_f_delta_ptx_vector_form.cu');
function_handle.GridSize(1) = ceil(num_data/function_handle.MaxThreadsPerBlock);
num_thread = min([num_data function_handle.MaxThreadsPerBlock]);
function_handle.ThreadBlockSize(1) = num_thread;

%%
%-----------------------------------------------------
% 1. Algorithm Parameters that you may want to adjust
%-----------------------------------------------------
load initial_condition;

dt=0.1;

% number of samples in one Period
num_steps = 1;
num_delta_sample = num_data;%1024*10;
                % lower with less computation but higher risk to be failed
                % higher with less risk but more computational burden

```

```
safety_factor_max = 1.00 + 1e-9;
safety_factor_min = 1.00 + 1e-9;
                % always greater than or equal to 1
                % smaller would give tighter upper bounds with higher risk to be
                % failed
                % larger would have less possbility to be failed but less
                % tight upper bounds

s_cale = 2;

% uncertain parameters dimension
delta_dim = 227;  % how many uncertain parameters
state_dim = 504;  % dimension of state vector x
which_x = 1;    % which x's bounds to be calculated

% Range of k, where k for k*delta_c
box_init = [1e-20 1e20];
                % you should know roughy where the k_optimal would be.
                % i.e, here I am quite sure that  1e-2 < f < 1e12
                % if your calculation failed, then you may adjust this

% ------------------------------------------
% 2. the function f(x,p) related setting
% ------------------------------------------
% what k-th step value bounds for x_k = f(x_k-1, p)
k_step_all = 1:300;
reset_steps = 3000;

% parameter bounds:
get_p_nominal; % p_centre from here
w_p = p_centre*0.1;

% state bounds
w_x = ((double(x_initial==0)+x_initial)*1e-6)*0;
w_x(w_x>1e-6)=1e-6*0;
x_centre = x_initial;

w_x_prev = w_x;
x_centre_prev = x_centre;

w_x_org = w_x;
x_centre_org = x_centre;

%-------------------------------------------------------------
% 3. Finally, you may want to use your own f(x,p) function.
% Then, update "get_f_delta.m"
%-------------------------------------------------------------

%%
%==================================================
% No need to change from here
%==================================================
for k_step = k_step_all

    %-----------------------------------
    % reset uncertain parameter range
    %-----------------------------------
    if (mod(k_step,reset_steps)==1) && (k_step > 1)

        x_centre_t(which_x) = (x_upper_max_all(end) + x_upper_min_all(end))/2;
        w_x_t(which_x) = (x_upper_max_all(end) - x_upper_min_all(end))/2;

```

```
99          x_remain=setdiff(1:state_dim,which_x);
100
101         for x_step=1:length(x_remain)
102
103             current_x = x_remain(x_step);
104
105             [x_upper_t,t1,t2] = cal_state_bounds_gpu(reset_steps, ...
106                 box_init,num_steps,num_delta_sample,delta_dim,state_dim, ...
107                 current_x,p_centre,w_p,x_centre_prev,w_x_prev, ...
108         safety_factor,reset_steps*2,[],dt);
109             c_max_t = 2*x_upper_t; c_min_t = -2*x_upper_t;
110
111             [x_upper_max_t,t1,t2] = cal_state_bounds_gpu(reset_steps, ...
112                 box_init,num_steps,num_delta_sample,delta_dim,state_dim, ...
113                 current_x,p_centre,w_p,x_centre_prev,w_x_prev, ...
114         safety_factor,reset_steps*2,c_max_t,dt);
115             x_upper_max_t = x_upper_max_t - c_max_t;
116
117             [x_upper_min_t,t1,t2] = cal_state_bounds_gpu(reset_steps, ...
118                 box_init,num_steps,num_delta_sample,delta_dim,state_dim, ...
119                 current_x,p_centre,w_p,x_centre_prev,w_x_prev, ...
120         safety_factor,reset_steps*2,c_min_t,dt);
121             x_upper_min_t = -x_upper_min_t - c_min_t;
122
123             x_centre_t(current_x) = (x_upper_max_t+x_upper_min_t)/2;
124             w_x_t(current_x) = (x_upper_max_t-x_upper_min_t)/2;
125
126         end
127
128         x_centre = x_centre_t(:);
129         x_centre_prev = x_centre;
130         w_x = w_x_t(:);
131         w_x_prev = w_x;
132
133     end
134
135     %-------------------------
136     % Calculate Bounds
137     %-------------------------
138     % calculate c_max and c_min
139     fprintf('=============================================\n');
140     fprintf('       Calculate c_max and c_min\n');
141     fprintf('=============================================\n');
142     [x_upper, x_lower, f_min_max_1] = cal_state_bounds_gpu(k_step, ...
143         box_init,num_steps,num_delta_sample,delta_dim,state_dim, ...
144         which_x,p_centre,w_p,x_centre,w_x,safety_factor,reset_steps,[],dt);
145     c_max = s_scale*x_upper;  % to just make sure all positive so it's doubled
146     c_min = -s_scale*x_upper; % for the similar reason, make it all negative
147
148     % calcualte upper and lower bounds for the maximum
149     fprintf('=============================================\n');
150     fprintf('       Calculate bounds for maximum\n');
151     fprintf('=============================================\n');
152     [x_upper_max, x_lower_max, f_min_max_2]  = cal_state_bounds_gpu(k_step, ...
153         box_init,num_steps,num_delta_sample,delta_dim,state_dim, ...
154         which_x,p_centre,w_p,x_centre,w_x,safety_factor_max,reset_steps,c_max,dt);
155     x_upper_max = x_upper_max - c_max;
156     x_lower_max = x_lower_max - c_max;
157
158     % calculate upper and lower bounds for the minimum
159     fprintf('=============================================\n');
160     fprintf('       Calculate bounds for minimum\n');
```

```
161     fprintf('=============================================\n');
162     [x_upper_min, x_lower_min, f_min_max_3] = cal_state_bounds_gpu(k_step, ...
163         box_init,num_steps,num_delta_sample,delta_dim,state_dim, ...
164         which_x,p_centre,w_p,x_centre,w_x,safety_factor_min,reset_steps,c_min,dt);
165     x_upper_min = -x_upper_min - c_min;
166     x_lower_min = -x_lower_min - c_min;
167 
168     f_min_max_by_simulation = [f_min_max_1; f_min_max_2; f_min_max_3];
169     f_min_max_by_simulation = ...
170     [min(f_min_max_by_simulation(:,1)) max(f_min_max_by_simulation(:,2))];
171 
172     x_upper_max_all(k_step) = x_upper_max;
173     x_lower_max_all(k_step) = x_lower_max;
174 
175     x_upper_min_all(k_step) = x_upper_min;
176     x_lower_min_all(k_step) = x_lower_min;
177 
178 end
179 
180 figure;
181 % maximum
182 k_step_all = [0 k_step_all];
183 plot(k_step_all*dt, [x_centre_org(which_x)+w_x_org(which_x) x_upper_max_all],'ro-');
184 hold on;
185 
186 % minimum
187 plot(k_step_all*dt, [x_centre_org(which_x)-w_x_org(which_x) x_upper_min_all],'bx-');
188 xlabel('k');
189 ylabel_str = sprintf('State x%d',which_x);
190 ylabel(ylabel_str);
```

B.2 cal_state_bounds_gpu.m

```
%-------------------------------------------------------------------------
% States bound calculation for discrete nonlinear systems using mu-bounds
%-------------------------------------------------------------------------
function [x_upper, x_lower, f_min_max] = cal_state_bounds_gpu(k_step, box_init, ...
    num_steps, num_delta_sample,delta_dim,state_dim,which_x, ...
    p_centre,w_p,x_centre,w_x,safety_factor,reset_steps,c_min_max,dt)
% Input:
%   k_step: bounds for k_step value of f(x,d)
%
%   box_init: initial search range of k,  k is searched in the range of
%       box_init(1) <= k <= box_init(2)
%
%   num_delta_sample: number of random samples along the 1-norm box edge
%
%   delta_dim: how many uncertain parameters
%
%   state_dim: dimension of state vector x
%
%   which_x: which x's bounds to be calculated, range = [1 state_dim]
%
%   p_centre: vector for the centres of uncertain parameters
%
%   w_p: weighting vector for the uncertain parameters so that p_delta is
%       between -1 and 1
%
%   x_centre: vector for the centres of state vector
%
%   w_x: weighting vector for the state vector so that x_delta is
%       between -1 and 1
%
%   safety_factor: safety factor for the upper bound, greater than 1,
%       smaller value may give more tighter bound but with higher risk
%       that may be wrong, this only applies to the upper bound
%
%   c_min_max: adjusting value for obtain bounds for max(f) and min(f)
%
% Output:
%   x_upper: upper bound
%   x_lower: lower bound
%   f_min_max: minimum and maximum values of f found during the sampling

% Programmed by Jongrae <Jongrae.Kim@glasgow.ac.uk>
% University of Glasgow
% Biomedical Engineering
% RobustLab: http://www.robustlab.org

%%
if nargin < 13
    reset_steps = 1000;
end
if nargin < 14
    c_min_max = 0;
end
if nargin < 15
    dt = 1;
end

if isempty(c_min_max)
```

```
59      c_min_max = 0;
60  end
61
62  tol = 1e-6;
63  current_big_box = box_init(2);
64  current_small_box = box_init(1);
65  f_min_max = [inf -inf];
66
67  %%
68  while abs(current_big_box-current_small_box) > tol
69
70      current_box_size = (current_big_box + current_small_box)/2;
71      current_k = current_box_size;
72
73      comp_pct = abs(current_big_box-current_small_box);
74
75      if comp_pct < 10 && comp_pct > 1e-3
76          fprintf('(%dth-step) Remaining calulation = %4.3f%% (a.u.)\n',k_step,comp_pct);
77      end
78
79      for idx=1:num_steps
80
81          % sample in the 1-norm box random face
82          delta_px = 2*rand(num_delta_sample,delta_dim+state_dim)-1;
83          rand_face = randi(delta_dim+state_dim,num_delta_sample,1);
84          idx_face = ...
85       sub2ind([num_delta_sample delta_dim+state_dim],(1:num_delta_sample)',rand_face);
86
87          delta_px(idx_face) = sign(rand(num_delta_sample,1)-0.5);
88          k_delta_c = current_k;
89
90          [f_real, f_delta] = I_ND_gpu_vector(k_step,delta_px,k_delta_c, ...
91              which_x,p_centre,w_p,x_centre,w_x,c_min_max,reset_steps,delta_dim,dt);
92
93          [f_real, idx_f_real_min] = min(f_real);
94          f_delta_min = min(f_delta,[],1);
95          f_delta_max = max(f_delta,[],1);
96
97          if f_delta_min(which_x) < f_min_max(1)
98              f_min_max(1) = f_delta(which_x);
99          end
100
101         if f_delta_max(which_x) > f_min_max(2)
102             f_min_max(2) = f_delta(which_x);
103         end
104
105
106         if f_real < 0
107             current_big_box = current_box_size;
108             x_lower = 1/current_k;
109             current_k_upper = current_k;
110             current_delta_px_worst = delta_px(idx_f_real_min,:);
111             break;
112         end
113
114     end
115
116     if (idx==num_steps) && (f_real > 0)
117         current_small_box = current_box_size;
118         x_upper = 1/current_k;
119     end
120
```

```
121
122 end % of while
123
124 % Refining the x_upper, i.e., k_lower
125 extreme_delta = sign(current_delta_px_worst);
126 upper_bd_not_converge = true;
127
128 epsilon_box = min([tol*10 norm(extreme_delta-current_delta_px_worst)]);
129 k_box = [current_k_upper/10 current_k_upper];
130
131 while upper_bd_not_converge
132
133
134     for idx=1:num_steps
135
136         % sample in the epsilon-norm box random face
137         delta_px = 2*rand(num_delta_sample,delta_dim+state_dim,1)-1;
138         delta_px = epsilon_box*delta_px;
139         delta_px = kron(ones(num_delta_sample,1),current_delta_px_worst) + delta_px;
140
141         k_delta_c = k_box(1) + abs(diff(k_box))/2;
142
143         [f_real, f_delta] = I_ND_gpu_vector(k_step, delta_px, ...
144             k_delta_c,which_x,p_centre,w_p,x_centre,w_x,c_min_max,reset_steps,delta_dim,dt);
145
146         f_real = min(f_real);
147         f_delta_min = min(f_delta,[],1);
148         f_delta_max = max(f_delta,[],1);
149
150         if f_delta_min(which_x) < f_min_max(1)
151             f_min_max(1) = f_delta(which_x);
152         end
153         if f_delta_max(which_x) > f_min_max(2)
154             f_min_max(2) = f_delta(which_x);
155         end
156
157         if f_real < 0
158             k_box(2) = k_delta_c;
159             break;
160         end
161
162     end
163
164     if ((idx==num_steps) && (f_real > 0))% || (abs(diff(k_box)) < tol)
165         upper_bd_not_converge = false;
166         x_upper = 1/k_box(2);
167     end
168
169
170 end % of while
171
172 %%
173 x_upper = safety_factor*x_upper;
```

B.3 I_ND_gpu_vector.m

```
function [f_real, f_delta] = I_ND_gpu_vector(k_step, delta_px, k_delta_c, ...
    which_x, p_centre, w_p, x_centre, w_x, c_min_max, reset_steps, num_delta,dt)
% Input:
%   k_step: bounds for k_step value of f(x,d)
%
%   delta_px: vector including delta_p and delta_x
%       e.g. delta_px = [ delta_p(1); delta_p(2); delta_x1; delta_x2]
%
%   k_delta_c: current k*delta_c
%
%   which_x: which x's bounds to be calculated, range = [1 state_dim]
%
%   p_centre: vector for the centres of uncertain parameters
%
%   w_p: weighting vector for the uncertain parameters so that p_delta is
%       between -1 and 1
%
%   x_centre: vector for the centres of state vector
%
%   w_x: weighting vector for the state vector so that x_delta is
%       between -1 and 1
%
%   c_min_max: adjusting value for obtain bounds for max(f) and min(f)
%
% Output:
%   f_real: real part of det(I-N Delta)
%   f_delta: f evaluated at the sampled point

% Programmed by Jongrae <Jongrae.Kim@glasgow.ac.uk>
% University of Glasgow
% Biomedical Engineering
% RobustLab: http://www.robustlab.org

global function_handle;

k_step = mod(k_step,reset_steps);
if k_step==0
    k_step=reset_steps;
end

%% CHANGE BELOW
%----------
num_state = length(x_centre);
num_data = size(delta_px,1);

% output state
switch which_x
    case 1
        ErbB1 = [59   60    73    75    76    77    78    79    80    81
82    83    84    85      86    87    88    89   235   236   237   238
239   240   241   242   244   245     246   247   248   249   250   251
359       360   361   362   365   366   368   369 371      372   388   412
441   442   443   463];
        ErbB12 = [47    48    90    91    96    97   102 103   108   109
114   115   120   121 126   127   132   133   213   216   219 222   225
228   231   234   262   263   279   282   295   298   308   311   375
376   389   416   435   444   445   464];
        ErbB13 = [ 49    50    92    93    98    99   104   105   110
```

```
    111   116   117   122   123   128   129   134   135   212   215   218 221
    224   227   230   233   264   265   280   283   296   299   309   312
    377   378   390   410   436   446   447   459 465   477 ];
52          ErbB14 = [ 51    52    94    95   100   101   106   107   112
    113   118   119   124   125   130   131   136   137   211   214   217 220
    223   226   229   232   266   267   281   284   297   300   310   313
    373   374   391   411   437   448   449   458  466   478];
53          out_idx = [ErbB1 ErbB1 ErbB12 ErbB13 ErbB14]; % ErbB1 is dimer
54      case 2
55          Erk = [345   346   348   349   359   360   361   362   363   364
    442   443   444   445   446   447   448   449   450   451   452 453   454
    455];
56          out_idx = Erk;
57      case 3
58          Akt = [405   407   470   471];
59          out_idx = Akt;
60      otherwise
61          out_idx = [];
62  end
63
64  % if the centre of x is zero, then take the absolute value of all
65  % perturbation
66
67  big_w_p = kron(ones(num_data,1),w_p(:)');
68  %big_p  = kron(ones(num_data,1),p_centre(:)') +
69  big_p =  big_w_p.*delta_px(:,1:num_delta);
70  big_w_x = kron(ones(num_data,1),w_x(:)');
71  big_x_c = kron(ones(num_data,1),x_centre(:)');
72
73  % take absolute perturbation if x_c = 0
74  zero_idx = (big_x_c==0)*(-1)+(big_x_c~=0);
75  x_state = big_x_c + big_w_x.*(delta_px(:,num_delta+1:end).*zero_idx);
76  x_state(x_state<0) = 0;
77
78  %keyboard
79
80  % include nominal for safety
81  x_state(end,:) = x_centre(:)';
82  big_p(end,:)=big_p(end,:)*0;
83
84  p_gpu = gpuArray(big_p(:));
85  x_state_gpu = gpuArray(x_state(:));
86  k_step_gpu = gpuArray(k_step);
87  num_data_gpu = gpuArray(num_data);
88
89  %dt = dt*0.001;
90  dt_gpu = gpuArray(dt);
91
92  x_out = zeros(size(x_state));
93  x_gpu_out = gpuArray(x_out);
94
95  f0 = zeros(size(x_state));
96  f0_gpu = gpuArray(f0);
97  f1 = zeros(size(x_state));
98  f1_gpu = gpuArray(f1);;
99  x0 = zeros(size(x_state));
100 x0_gpu = gpuArray(x0);
101 x1 = zeros(size(x_state));
102 x1_gpu = gpuArray(x1);
103
104 %-------------------------------------------------
105 % ptx option: !nvcc -ptx get_f_delta_ptx.cu
```

```
106 %-------------------------------------------------------
107 x_gpu_out = ...
108 feval(function_handle, ...
109       x_gpu_out, p_gpu, x_state_gpu, k_step_gpu, ...
110       num_data_gpu, dt_gpu, f0_gpu, f1_gpu, x0_gpu, x1_gpu);
111 x_state = gather(x_gpu_out);
112 x_state = reshape(x_state, num_data, num_state);
113 f_delta = sum(x_state(:,out_idx),2);
114
115 x_state_gpu = gpuArray(x_state(:));
116
117 f_real = 1-k_delta_c*abs(f_delta(:)+c_min_max);
118
119
120 end
```

B.4 get_f_delta_ptx_vector_form.cu

```
// Programmed by Jongrae Kim <Jongrae.Kim@glasgow.ac.uk>

// Warning
// 1. Do not use a variable as an input and output at the same time

#define NUM_STATE   504
#define NUM_DELTA   227

// function header
__device__ void f_p_x(double * x_out,
            const double * p_in, const double * x_in, const int num_data, const int idx);

// main function
__global__ void get_f_delta(    double * x2,
                        const double * p_in, const double * x_in,
                const int k_step, const int num_data, const double dt,
                double * f0, double * f1, double * x0, double * x1)
{
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    int current_time_step;
    int current_state_step;

    // integration using Two-Step Adams-Bashforth
    for (current_time_step=0; current_time_step < k_step; current_time_step++)
    {

      if (current_time_step == 0)
      {
        // f0 = f(x0)
            f_p_x(f0, p_in, x_in, num_data, idx);

        // x2 = x0 + dt f(x0): special case for current_time_step = 0
            for (current_state_step=0; current_state_step < NUM_STATE; current_state_step++) {
                x2[idx+current_state_step*num_data] = x_in[idx+current_state_step*num_data]
                + dt*f0[idx+current_state_step*num_data];

            if (x2[idx+current_state_step*num_data] < 0.0)
                x2[idx+current_state_step*num_data] = 0.0;

            //save for the next step
            x0[idx+current_state_step*num_data] = x_in[idx+current_state_step*num_data];
            x1[idx+current_state_step*num_data] = x2[idx+current_state_step*num_data];
        }
      }
      else
      {
        //-----------------------------------------
        // Two-step Adams-Bashforth Integration
        //-----------------------------------------

        // f(x0) & f(x1)
            f_p_x(f0, p_in, x0, num_data, idx);
        f_p_x(f1, p_in, x1, num_data, idx);

        // x_2 =  x1 + 1.5*dt*f(x1) - 0.5*dt*f(x0)
            for (current_state_step=0; current_state_step < NUM_STATE; current_state_step++) {
            x2[idx+current_state_step*num_data] = x1[idx+current_state_step*num_data]
                        + 1.5*dt*f1[idx+current_state_step*num_data]
```

```
                        - 0.5*dt*f0[idx+current_state_step*num_data];

            if (x2[idx+current_state_step*num_data] < 0.0)
                x2[idx+current_state_step*num_data] = 0.0;

            // save for the next step
            x0[idx+current_state_step*num_data] = x1[idx+current_state_step*num_data];
            x1[idx+current_state_step*num_data] = x2[idx+current_state_step*num_data];
        }

      }


    }

}


// function definition
__device__ void f_p_x(double * x_out,
            const double * p_in, const double * x_in, const int num_data, const int idx)
{

    // make the state index vector
    int current_state_step;
    int state_idx[NUM_STATE];
    for (current_state_step=0; current_state_step < NUM_STATE; current_state_step++)
        state_idx[current_state_step] = idx + current_state_step*num_data;

    // make the delta index vector
    int current_delta_step;
    int delta_idx[NUM_DELTA];
    for (current_delta_step=0; current_delta_step < NUM_DELTA; current_delta_step++)
        delta_idx[current_delta_step] = idx + current_delta_step*num_data;


    // kinetic parameters
    double   kd1=3.300000e-02 + p_in[delta_idx[0]];
    double   k1c=8.000000e+02 + p_in[delta_idx[1]];
    double   kd1c=1.000000e+00 + p_in[delta_idx[2]];
    double   kd1d=1.000000e-01 + p_in[delta_idx[3]];

    // omitted ...

    // differential equations
     x_out[state_idx[0]] = 0.0;
    // keep EGF constant, e.g. 5e-9 = 5 nM
    // EGF (See Figure 2 in the Reference) or 0.01e-9 (Figure 2)

     x_out[state_idx[1]] = -(k1*x_in[state_idx[0]]*x_in[state_idx[1]]-kd1*x_in[state_idx[2]]);
     x_out[state_idx[1]] = x_out[state_idx[1]]
            -(k6*x_in[state_idx[1]]-kd6*x_in[state_idx[242]]);
     x_out[state_idx[1]] = x_out[state_idx[1]]
            -(k120b*x_in[state_idx[482]]*x_in[state_idx[1]]-kd120*x_in[state_idx[60]]);
     x_out[state_idx[1]] = x_out[state_idx[1]]
            -(k120b*x_in[state_idx[483]]*x_in[state_idx[1]]-kd120*x_in[state_idx[62]]);
     x_out[state_idx[1]] = x_out[state_idx[1]]
            +(k122*x_in[state_idx[417]]*x_in[state_idx[35]]-kd122*x_in[state_idx[1]]);

     x_out[state_idx[2]] = (k1*x_in[state_idx[0]]*x_in[state_idx[1]]-kd1*x_in[state_idx[2]]);
     x_out[state_idx[2]] = x_out[state_idx[2]]
            -(k2*x_in[state_idx[2]]*x_in[state_idx[8]]-kd2*x_in[state_idx[9]]);
```

```
121       x_out[state_idx[2]] = x_out[state_idx[2]]
122             -(k2*x_in[state_idx[2]]*x_in[state_idx[2]]-kd2*x_in[state_idx[11]]);
123       x_out[state_idx[2]] = x_out[state_idx[2]]
124             -(k2b*x_in[state_idx[2]]*x_in[state_idx[14]]-kd2b*x_in[state_idx[15]]);
125       x_out[state_idx[2]] = x_out[state_idx[2]]
126             -(k2b*x_in[state_idx[2]]*x_in[state_idx[16]]-kd2b*x_in[state_idx[17]]);
127
128      // omitted ...
129
130       x_out[state_idx[503]]=(k122*x_in[state_idx[498]]*x_in[state_idx[35]]
131             -kd122*x_in[state_idx[503]]);
132       x_out[state_idx[503]] = x_out[state_idx[503]]
133             +(k123h*x_in[state_idx[243]]*x_in[state_idx[35]]-kd123h*x_in[state_idx[503]]);
134  }
```